UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Diseño de un proceso de gestión de infraestructura tecnológica y propuesta de un sistema que lo implemente

Autores: Leandro Di Tommaso y Joaquín Salvarredy
Director: Lic. Miguel Luengo
Codirector: Lic. Christian Rodriguez
Asesor profesional: Lic. Mariano Droz
Carrera: Licenciatura en Sistemas

Resumen

Tomando como caso de estudio a la Dirección General de Tecnologías de la Información y la Comunicación de la Universidad Nacional de Entre Ríos, se analiza su infraestructura tecnológica, relevando servicios y aplicaciones para proponer un proceso para la gestión de las mismas, estableciendo el ciclo de vida que deben tener dichas aplicaciones en testing.

Aplicando las metodologías ágiles a las áreas de administración de sistemas, surge el movimiento DevOps, que promueve la relación laboral colaborativa entre las áreas de desarrollo y sistemas, proponiendo una forma de trabajo para administrar la infraestructura como código.

La tesina propone el uso de herramientas de aprovisionamiento en el marco de DevOps con el fin de implementar una nueva arquitectura para varios de los sistemas del organismo, teniendo en cuenta diferentes factores, como la posibilidad de versionar los cambios, gestionar los diferentes entornos y automatizar el despliegue de sistemas y servicios.

Palabras Claves

DevOps – Ansible – Chef – Puppet – Docker – Vagrant – Kanban – Lean – Aprovisionamiento – Ambientes de desarrollo – Gestión de aplicaciones – Ciclo de vida – Testing – Producción – Gestión de DNS – Infraestructura como código – Microservicios – Sistemas inmutables – Metodologías ágiles – Automatización – GIT – Versionado de la infraestructura – Contenedores – LXC – OpenVZ

Trabajos Realizados

Análisis de la infraestructura de la DGTIC de la Universidad Nacional de Entre Ríos, rediseñando e implementando parte de la misma con la metodología DevOps, utilizando herramientas para la automatización de las instalaciones de servidores como lo son Ansible y Docker.

Se propone una gestión por procesos para los sistemas, el ciclo de vida de las aplicaciones en testing y cómo deben manejarse los cambios en sistemas en producción.

Conclusiones

Mejora en la gestión de proyectos haciendo uso de la metodología Kanban.
La adopción de DevOps tiene como ventaja tratar a la infraestructura como código, la cual se puede versionar y replicar de manera fácil, mejorando la comunicación entre los grupos de desarrollo y operaciones.
Herramientas como Ansible facilitan la tarea de automatizar y aprovisionar cientos de servidores, mejorando sustancialmente la tarea de los administradores de sistemas.

Trabajos Futuros

Analizar herramientas para manejar la infraestructura de contenedores con Docker.
Proponer una política de gestión para las actualizaciones de las aplicaciones.
Monitoreo y estadísticas de una infraestructura como código.
Restauración masiva de la infraestructura independientemente de la plataforma.
Extender el aprovisionamiento automatizado a la gestión del equipamiento de red.

Fecha de la presentación: Noviembre 2015

Universidad Nacional de La Plata

Facultad de Informática

Tesina de Licenciatura

Diseño de un proceso de gestión de infraestructura tecnológica y propuesta de un sistema que lo implemente

Di Tommaso, Leandro Damián
Salvarredy, Joaquín Gonzalo

Director: Lic. Miguel Angel Luengo
Co-Director: Lic. Christian Rodriguez
Asesor Profesional: Lic. Mariano Droz

6 de noviembre de 2015

Agradecimientos

A mi familia y amigos, porque esta tesina nunca hubiera sido posible sin el apoyo incondicional que me brindaron siempre. A Miguel y a Christian, por su tiempo, por la ayuda constante y porque supieron motivarme para enfrentar este desafío. A Ana, por su alegría, su paciencia y por su aporte creativo. A Joaquín, porque incluso a la distancia, trabajar con él fue como estar jugando. A aquel maravilloso grupo de soporte del CeSPI, con el que crecí y al que le debo mucho de lo que soy. A mis compañeros de trabajo, porque de todos he aprendido algo y a varios de ellos hoy puedo llamarlos amigos. A la Universidad Nacional de Entre Ríos y a Mariano, por habernos dado el marco para llevar adelante esta tesina. A la Universidad Nacional de La Plata, porque se transformó en mi segundo hogar. A mi país, Argentina, por haberme dado la oportunidad de estudiar y de formarme profesionalmente.

Leandro Di Tommaso

A Miguel y a Christian por su motivación, insistencia, predisposición y dedicación. A Mariano por facilitarnos el lugar para desarrollar el trabajo y sus innumerables aportes. A mis viejos compañeros del CeSPI, por los buenos momentos y el desarrollo profesional durante tantos años. A mis nuevos compañeros de la UNER, que han colaborado para que este trabajo salga adelante. A Leo, por el reencuentro y hacerme sentir como si todavía estuviera en La Plata. A mi familia y amigos, por su apoyo incondicional. A la Universidad Nacional de La Plata, por sus profesores, ayudantes y compañeros que me formaron, por esos lindos años vivido y sobre todo, por los amigos que fui sumando en el camino.

Joaquín Salvarredy

Índice

Introducción **6**

1. Capítulo I **11**
1.1. Objetivos . . . 11
1.2. Contexto del trabajo . . . 11
1.2.1. Relevamiento del caso de estudio . . . 12
1.2.2. Ambientes de desarrollo . . . 13
1.2.3. Respaldo de la infraestructura . . . 14
1.2.4. Aulas Virtuales . . . 15
1.2.5. Sistemas *SIU* . . . 15

2. Capítulo II **16**
2.1. Antecedentes . . . 16
2.1.1. Flickr . . . 16
2.1.2. PAMI . . . 17
2.1.3. Facebook . . . 17
2.1.4. Automated Labs . . . 17
2.2. Bases teóricas . . . 18
2.2.1. Metodologías ágiles . . . 18
2.2.2. Lean Software . . . 19
2.2.3. Waste . . . 20
2.2.4. Kanban . . . 21
2.2.5. Teoría de las Restricciones . . . 22
2.2.6. DevOps . . . 22
2.2.7. Sistemas Inmutables . . . 25

3. Capítulo III **27**
3.1. Definiciones . . . 27
3.1.1. Ambientes . . . 27
3.2. Herramientas a utilizar . . . 27
3.2.1. Gestión de proyectos . . . 27
3.2.2. Herramienta de automatización . . . 28
3.3. Gestión de aplicaciones . . . 29
3.3.1. Proceso de solicitud de nueva aplicación . . . 29
3.3.2. Reempadronamiento de aplicaciones . . . 31
3.3.3. Gestión de aplicaciones en testing . . . 32
3.3.3.1. Ciclo de vida . . . 33
3.3.3.2. Proceso de gestión de la caducidad . . . 33

3.3.4. Proceso de gestión de cambios en producción 36
3.4. Aulas Virtuales 37
3.4.1. Diseño de arquitectura 38
3.4.2. Implementación 39
3.4.3. Moodle 2.0 43
3.5. Sistemas *SIU* 44
3.5.1. Diseño de arquitectura 45
3.5.2. Implementación 46
3.5.3. Ambientes de desarrollo y testing 48
3.6. Casos de análisis e implementación 51
3.6.1. Alojamiento web 51
3.6.2. Servicio de DNS 54
3.6.2.1. Proceso de gestión del DNS 55
3.6.2.2. Gestionando el DNS con *Ansible* 57

4. Capítulo IV **61**
4.1. Análisis 61
4.1.1. El problema de las metodologías ágiles 61
4.1.2. Kanban en el área de operaciones 63
4.1.3. DevOps y la infraestructura como código 64
4.1.4. Información y persistencia 66
4.1.5. Estandarización y gestión de cambios 68
4.2. Limitaciones 68
4.2.1. Restauración masiva de la infraestructura 68
4.3. Conclusiones 69
4.3.1. Aportes realizados 70
4.3.2. Aprendizajes adquiridos 73
4.4. Trabajos futuros 73
4.4.1. Monitoreo 74
4.4.2. Estadísticas 74
4.4.3. Gestión de las actualizaciones 75
4.4.4. Infraestructura con Docker 75
4.4.5. Restauración masiva de la infraestructura 76
4.4.6. Aprovisionamiento de la red 77

A. ANEXO I **78**
A.1. Tecnologías de Virtualización 78
A.1.1. VMware 78
A.1.2. VirtualBox 78
A.1.3. KVM 79
A.2. Vagrant 79

A.2.1. Aprovisionamiento 80
A.2.2. Generadores de *Vagrantfile* 81
A.3. Contenedores 81
A.3.1. chroot 82
A.3.2. OpenVZ 83
A.3.3. Linux Containers: LXC 83
A.3.4. Docker 86
A.3.4.1. Imágenes 87
A.3.4.2. Registros 87
A.3.4.3. Contenedores 87
A.3.4.4. Cómo funciona Docker 88
A.3.4.5. Dockerfile 89

B. ANEXO II **91**
B.1. Ansible 91
B.1.1. Playbooks 91
B.1.2. Hosts 92
B.1.3. Lista de *Tasks* 92
B.1.4. Roles 93
B.2. Puppet 93
B.2.1. Arquitectura Maestro y Agentes 94
B.2.2. Puppet Enterprise (PE) 94
B.2.3. Módulos 95
B.3. Chef 95
B.4. SaltStack 98
B.4.1. Componentes 98

Índice de figuras

1. Gestionando proyectos con la metodología *Kanban* 28
2. Solicitud de nueva aplicación 30
3. Ciclo de vida de una aplicación en testing 33
4. Proceso de gestión de aplicación en testing 34
5. Gestión de cambios en aplicaciones en producción 36
6. Arquitectura de la Infraestructura del Moodle 38
7. Arquitectura de la Infraestructura de los sistemas *SIU* 46
8. Arquitectura *Docker* de Web Hosting 53
9. Proceso de gestión del DNS 56
10. Entrada de DNS en *Gitlab* 59
11. Flujo de trabajo tradicional entre desarrollo y operaciones . . 61
12. Metodologías ágiles en el desarrollo, antes de DevOps 62
13. Flujo de trabajo con DevOps 63

Introducción

La tarea de instalar y configurar un servicio para luego ponerlo operativo es, en la mayoría de los casos, una tarea artesanal que está directamente asociada al administrador, quedando a total criterio de este. Así, en una misma organización, es posible encontrar servidores que cumplen el mismo rol, configurados por personas diferentes y de manera distinta, haciendo más difícil su administración. Otro problema muy común es olvidar instalar algún servicio adicional relacionado con la gestión, como puede ser un agente necesario para el monitoreo o para realizar las copias de seguridad. Y el principal problema en estos casos es que estos olvidos se evidencian en el momento en que surge un problema.

A la hora de realizar el mantenimiento de los equipos, ya sea por una actualización de servicios, modificaciones por ajustes o para resolver inconvenientes, queda librado a la voluntad del administrador tomar los recaudos del caso, como puede ser realizar una copia de los archivos de configuraciones anteriores, necesarios en caso de tener que restaurar el servicio.

Suele ocurrir también que, el hecho de tener que replicar un servicio, particularmente en instalaciones nuevas, implica muchas veces apelar a la memoria para articular las piezas que hacen funcionar el esquema completo de dicho servicio. En otras situaciones, si se tiene un servidor con el servicio corriendo se suele recurrir a copiar archivos de configuración, lo que en la mayoría de los casos origina problemas si las versiones del software utilizado no son las mismas, dado que entre versiones las opciones de configuración suelen cambiar y no ser totalmente compatibles.

El peor escenario incluso es cuando un servidor deja de funcionar y no puede accederse al contenido de su disco. En estas situaciones, sobre todo si se trata de un servidor crítico, restaurar el mismo es una tarea que puede llevar muchas horas, regidas por la tensión y el tiempo y que implica un trabajo normalmente sin descanso, debido a la demanda para que el servicio vuelva a estar en línea en el menor plazo posible. La consecuencia, más allá de la prolongada interrupción del servicio, es que la probabilidad de que queden aspectos mal configurados o incompletos es muy alta, lo que puede dar como resultado una prestación inestable o, incluso, terminar en pérdida de información.

Una solución que buscan los administradores de sistemas ante esta situación es agregar a la tradicional copia de seguridad de los datos, un resguardo también de los archivos de configuración. Esta estrategia, aunque efectiva

en muchas situaciones, tiene sus propios inconvenientes. En primer lugar, no resulta natural hacer copia de seguridad de configuraciones, debido a que se mezclan datos propios de las aplicaciones con archivos de configuración del sistema. Resguardar además datos de sistemas y archivos de configuración del servidor implica prácticamente hacer copias del sistema operativo completo. Existe además otro problema y es la frecuencia de los cambios de las configuraciones respecto de la frecuencia de las copias de resguardo. Mientras que las configuraciones suelen permanecer estáticas durante mucho tiempo, no es extraño que la modificación de las mismas se de varias veces en un mismo día. Difícilmente pueda una copia de seguridad del sistema asegurar un resguardo de cada uno de esos cambios.

En pos de evitar esos problemas, muchas organizaciones comenzaron a generar numerosos procedimientos y enormes volúmenes de documentación que les permitiera estandarizar la manera de gestionar su infraestructura, desde las configuraciones hasta la forma de introducir los cambios, no sólo para facilitar su predictibilidad y comprensión sino también para poder replicarla en cualquier momento. Dos claros ejemplo de esto son los procesos de gestión de configuraciones y de gestión de cambios, ambos de *ITIL*.

En la práctica, administrar toda la documentación de la infraestructura se transformó en un nuevo problema y dicha documentación terminó resultando de poco valor en muchos casos. Se dice esto porque mantenerla se vuelve extremadamente complejo por el carácter netamente dinámico de la infraestructura, que provoca que la documentación se vuelva obsoleta rápidamente. Con el paso del tiempo, esta desactualización implica que no sea posible confiar en los documentos para realizar ninguna tarea, no cumpliendo el objetivo para el que fueron concebidos. Por otro lado, el esfuerzo necesario para mantener actualizada la documentación da como resultado una significativa pérdida de productividad y aumenta los costos de la organización.

La "llegada" de la virtualización, o mejor dicho, el uso masivo de la misma y su popularización, desencadenó en el surgimiento de múltiples herramientas, solucionando varios de los aspectos mencionados, como la posibilidad de disponer más rápidamente de nuevos servidores, disminuir los tiempos necesarios para restaurar un servicio o instalar uno nuevo, entre otros. Pero sin dudas, dos características de suma importancia a destacar son la posibilidad de utilizar plantillas (*templates*) de equipos y la de realizar instantáneas (*snapshot*) de los servidores completos.

De repente, los administradores de sistemas creían que habían encontrado la solución a todos sus problemas, aunque con el trabajo diario fueron descubriendo que, si bien se simplificaron varios procesos, se redujeron los

tiempos y se aumentó la confiabilidad y disponibilidad de los servicios, también se introdujeron nuevos inconvenientes que antes no existían y que aún muchos de esos históricos problemas seguían sin obtener una solución real y definitiva.

Por un lado, proliferaron rápidamente las plantillas de los sistemas, llegando a tener tantas que, con el tiempo, se volvió difícil determinar cuál era la apropiada para usar ante un caso de necesidad. Sumado a esto, las plantillas también suelen volverse obsoletas para cuando se las necesita, exigiendo su actualización periódica (tarea que se realiza al momento de necesitarlas, perdiendo entonces esa ventaja de tener en segundos un sistema listo para ser usado). Por otro lado, los *snapshots* implican la copia de todo un sistema completo con su estado exacto, algo que puede ser indeseable en algunas situaciones. Si se imagina, por ejemplo, un servidor de correo que tiene quinientos correos encolados, listos para ser enviados y del que, en ese momento, se toma una instantánea; los correos serán enviados normalmente pero si más tarde se requiere restaurar esa instantánea, volverán a enviarse los mismos correos que estaban encolados al momento de ser generada, lo que puede causar un gran problema. Sumado a esto, las instantáneas copian el sistema completo, incluyendo datos; esto es una mala forma de hacer una copia de seguridad de los datos y es también una mala forma de resguardar un sistema, debido a que las copias de seguridad en la mayoría de los casos no son incrementales, utilizando demasiado espacio de almacenamiento. Esto implica, a su vez, reducir la cantidad de copias de seguridad que es posible almacenar, teniendo menos "historia" de la que disponer.

Por otro lado, si un sistema fue comprometido y se descubre un tiempo después, quizá ya no se tenga la copia del sistema antes de haber sido comprometido. Y si se tuviera, en esa copia los datos serían viejos, lo que implicaría combinar ambas instantáneas. En fin, son muchos los ejemplos que pueden encontrarse, alcanza con que el lector pueda ver alguno de ellos para comprender el punto de esta argumentación.

Como se menciona anteriormente, la virtualización introdujo además problemas nuevos, siendo sin dudas el más importante de ellos la especialización de servidores, antes físicos, ahora virtuales. En los tiempos en los que los servicios se implementaban directamente sobre el equipo físico era común encontrarse con un servidor que cumplía múltiples funciones, como por ejemplo, brindar servicio web, de correo y de DNS. Esto se daba de esta manera por cuestiones de disponibilidad de hardware, espacio físico, instalaciones eléctricas, refrigeración, entre otros factores. Con la virtualización, se volvió natural tener al menos tres servidores independientes cumpliendo esos roles (se pue-

de decir al menos porque el correo o el servicio web también comenzaron a explotarse aún más). La ventaja de la seguridad y la independencia de los servicios, de la mano de la simplicidad de disponer de servidores virtuales, fue y es la principal causa de esta separación. Una de las más importantes consecuencias negativas de este enfoque es que la cantidad de servidores a administrar creció casi exponencialmente, y dónde antes existían diez equipos pasó a ser común encontrarse con más de cuarenta. Puede verse fácilmente cómo la tarea de gestionar los servidores se volvió más compleja. Sin ir más lejos, si se piensa por ejemplo en el surgimiento de una vulnerabilidad a nivel de sistema operativo, las acciones para solucionarlas implican trabajar equipo por equipo. Esto determinó incluso que fuera necesario disponer de más recursos humanos. Y con ellos, mayor variabilidad en la manera de instalar los equipos, introduciendo todavía más complejidad a la gestión.

Ahora bien, hasta ahora se hizo foco en los problemas internos propios del área de administración de sistemas, pero aún queda una problemática delicada y que suele ser factor de conflicto en toda organización y es la relación y la integración con otras áreas, particularmente con la de desarrollo de software. En este sentido, los problemas son de los más diversos pero siempre tienen algo en común: todos ellos generan, indefectiblemente, un conflicto.

Los desarrolladores de software suelen contar, en las máquinas donde realizan los desarrollos, con un entorno completo para programar y testear las aplicaciones, entorno que adaptan a su aplicación y aplicación que adaptan al entorno. Cuando la aplicación está lista para ser puesta en producción surge el primero de los problemas y es que, el entorno donde fue desarrollado el software nada tiene que ver con el entorno productivo. Surge un inconveniente difícil de resolver en esta instancia, debido a que el área de infraestructura suele negarse (y con justa razón) a instalar determinados paquetes en sus servidores productivos para que la aplicación funcione, mientras que el equipo de desarrolladores también suele negarse (con su propia justa razón) a modificar toda su aplicación para que se adapte el entorno de producción. Esto no sólo genera problemas en la relación entre las áreas sino que retrasa la implementación en producción de los sistemas, afectando a la organización completa.

La virtualización aportó lo suyo en este aspecto, debido a que en muchas organizaciones se empezó a adoptar como política que el área de infraestructura dé a los desarrolladores una máquina virtual con un entorno exactamente igual al de producción. Esto redujo en una gran proporción los problemas al momento de pasar el sistema a producción, sin eliminarlos, debido a que esas

máquinas virtuales con el tiempo dejan de ser esa copia exacta para volver a adaptarse a las necesidades del desarrollador.

Las organizaciones que por la posibilidad de contar con los recursos de hardware adicionales necesarios, disponen de un ambiente de pre-producción, minimizan el problema del pasaje al entorno productivo aunque, como en el caso anterior, no lo eliminan. Nuevamente, es muy difícil garantizar que los ambientes de pre-producción y de producción sean idénticos, lo que da como resultado un nuevo problema porque además, ahora, debe administrarse un conjunto adicional de servidores.

La otra gran problemática que existe en la relación entre las áreas de desarrollo de software y de infraestructura es que, por un lado, el área de desarrollo necesita una respuesta rápida y constante del área de infraestructura para poder llevar adelante su trabajo, pero el área de infraestructura tiene sus propios problemas y prioridades; para el área de infraestructura, el área de desarrollo es uno más de sus usuarios. Esto también genera conflictos y retrasos.

En muchas situaciones, para solucionar todos estos inconvenientes, minimizar la burocracia y, reducir la carga de trabajo impuesta desde el área de desarrollo sobre la de infraestructura, se termina en uno de los peores escenarios que es el de otorgar a los desarrolladores de software, que no son operadores de sistemas, acceso con privilegios administrativos a los servidores en producción.

Ante todo este planteo, queda claro que la evolución vertiginosa que ha tenido la tecnología en todo este tiempo poco ha hecho para solucionar los problemas que desde hace muchos años existen en la gestión de la infraestructura tecnológica de las organizaciones. Incluso, como se expuso en los párrafos anteriores, ha contribuido en muchas situaciones a aumentar los problemas.

Esta tesina parte de la base de considerar que las nuevas herramientas de automatización y aprovisionamiento de la infraestructura, junto con un importante cambio en la metodología de trabajo del área de operaciones, pueden constituir la clave para resolver los problemas presentados hasta aquí. Así es que en esas cuestiones se basará el desarrollo de esta investigación, en la que se analizará la problemática planteada, tomando como caso específico a la Dirección General de Tecnologías de la Información y la Comunicación (DGTIC) de la *Universidad Nacional de Entre Ríos*, proponiendo posibles soluciones e implementando algunas de dichas soluciones.

1. Capítulo I

1.1. Objetivos

El objetivo de esta tesina es trabajar sobre la infraestructura actual de la Dirección General de Tecnologías de la Información y la Comunicación (DGTIC) de la *Universidad Nacional de Entre Ríos* para lograr:

- Definir un nuevo proceso para la administración de su infraestructura.
- Utilizar herramientas de automatización para la gestión de los servidores.
- Adoptar la filosofía *DevOps*.

Con lo anterior, se propone:

- Estandarizar y optimizar la prestación de servicios.
- Maximizar la disponibilidad de los servicios y sistemas.
- Utilizar un sistema de control de versiones para las configuraciones de los servicios.
- Mejorar la estructura de servicios y el ciclo de vida.
- Minimizar los tiempos de respuesta y la capacidad de recuperación ante fallos.
- Disponer de la capacidad de replicar ambientes idénticos en pocos minutos.
- Facilitar la gestión de la infraestructura y de las máquinas virtuales.

1.2. Contexto del trabajo

Para comprender el presente trabajo es importante conocer el contexto en que se desarrolla el mismo. Por tal motivo, se presenta en esta sección la generalidad de la infraestructura con que cuenta la *DGTIC de la Universidad Nacional de Entre Ríos*, introduciendo la problemática actual y describiendo los principales sistemas y los ambientes existentes.

1.2.1. Relevamiento del caso de estudio

En la actualidad, la infraestructura de la *DGTIC*[1] de la *Universidad Nacional de Entre Ríos* tiene un total de sesenta servidores, de los cuales más de cincuenta están virtualizados utilizando la plataforma de *VMware*[2].

Los sistemas operativos de los servidores son bastante diversos y se resumen en el siguiente cuadro.

Sistema Operativo	**Versión**	**Cantidad**
CentOS	5.10	4
Debian	4.0	2
Debian	5.0	8
Debian	6.0	20
Debian	7.0	13
Pfsense		4
Ubuntu	12.04	4
Ubuntu	14.04	1
Otros		4

De los servicios que ejecutan los servidores listados, se relevaron solamente los servidores web y de bases de datos y se muestran debajo.

- Servidores web
  - *Apache*: cantidad 40
  - *Nginx*: cantidad 1
  - *Tomcat*: cantidad 5
- Servidores de base de datos:
  - *PostgreSQL*: cantidad 26
  - *MySQL*: cantidad 18

Con el breve relevamiento presentado, se pueden sacar varias conclusiones, algunas de las cuáles se listan a continuación.

[1]La Dirección General de TIC se encuentra en el Rectorado y brinda una amplitud de servicios a todas las unidades académicas.

[2]Utilizando VMware Center para una administración centralizada.

- La diversidad de los sistemas operativos con los que se trabaja hace que sea complejo mantener configuraciones unificadas.
- Es fácil notar que muchos servidores ejecutan versiones viejas de los sistemas operativos (algunas sin soporte desde hace tiempo). También se tienen versiones antiguas del software que presta los servicios e, incluso, los servidores no tienen aplicadas las actualizaciones correspondientes.
- Cada servidor de aplicaciones tiene su propio servicio web y de base de datos instalados, lo que implica un uso no optimizado de los recursos de hardware a la vez que aumenta la complejidad en la administración de los mismos.
- Al tener instalados demasiados servidores de bases de datos, cada uno independiente del otro, se dificulta la implementación de algún esquema de redundancia a nivel de servicio[3].

1.2.2. Ambientes de desarrollo

La *DGTIC de la Universidad Nacional de Entre Ríos* realiza desarrollos internos utilizando el *framework SIU-Toba*, el cuál es la base para los desarrollos implementados por el *Sistema de Información Universitaria* (*SIU*), como son los sistemas *Guaraní*, *Mapuche*, *Pilagá*, entre otros.

Para cumplir con sus funciones, los desarrolladores cuentan, en sus estaciones de trabajo, con instalaciones propias del *framework* que les permite desarrollar en forma local. Dichos estaciones de trabajo tienen sistemas operativos *Ubuntu* y/o *Windows* en diferentes versiones. Por otro lado, el servidor de producción que brinda los servicios utiliza como sistema operativo *Debian* en su versión *Squeeze* (6.0). Puede notarse así la heterogeneidad de los ambientes de desarrollo.

Esta manera de trabajar, sin ningún tipo de estandarización, genera grandes inconsistencias y dificultades al momento de realizar la puesta en producción de los sistemas desarrollados. En la presente tesina, entonces, se analizará esta problemática junto con posibles soluciones para dar como resultados procesos de trabajo y herramientas sugeridas que pueden simplificar la tarea de generar los ambientes de desarrollo y la puesta en producción de

[3]Tanto *PostgreSQL* como *MySQL* soportan esquemas de replicación *master-slave*, *master-master* y *multi-master*

los sistemas, así como la introducción de nuevos cambios, utilizando la estrategia del despliegue continuo.

1.2.3. Respaldo de la infraestructura

Se entiende en esta tesina como respaldo de la infraestructura al hecho de poder disponer de mecanismos que permitan restaurar el funcionamiento de cualquier servidor o servicio para evitar una reinstalación desde cero, con los consecuentes beneficios que ello conlleva:

- Menor tiempo de configuración y restauración.
- Eliminación de errores por descuidos y/o omisión.
- Simplificación en el proceso de restauración de la infraestructura.
- Independencia del operador que deba realizar la tarea de restauración.

Con la intención de contar con el respaldo mencionado, en la *DGTIC de la Universidad Nacional de Entre Ríos* se realiza semanalmente un *snapshot* completo de cada máquina virtual que está corriendo en la infraestructura, los cuales se copian a un almacenamiento secundario. Este mecanismo da la posibilidad de restaurar el estado de una máquina virtual pero teniendo varios inconvenientes asociados:

- Espacio ocupado en el almacenamiento secundario: toda la infraestructura en la actualidad tiene un tamaño total cercano a los dos terabytes.
- Tiempo de restauración: debido al tamaño de los *snapshots*, es necesario copiar muchos gigabytes de información por medio de la red de datos, luego de lo cuál se descomprime el *snapshot* que contiene la máquina virtual de interés. Este proceso, dependiendo del tamaño ocupado, puede demorar desde algunos minutos hasta varias horas.
- Rotación: por el espacio ocupado por cada *snapshot* se hace una rotación quincenal, lo que sólo brinda una historia muy breve que imposibilita acceder a versiones más antiguas del servidor.

1.2.4. Aulas Virtuales

La *DGTIC de la Universidad Nacional de Entre Ríos* dispone de una plataforma de aulas virtuales que cuenta con alrededor de 30.000 usuarios y unos 700 cursos. El software que se utiliza es Moodle[4] en la versión 1.9.19 montado sobre un servidor que tiene como sistema operativo Debian 4.0[5]. Por la cantidad de usuarios y la criticidad del servicio prestado, dicho servidor se ha transformado en lo que se denomina una *fragile box*[42]. A su vez, en paralelo se está testeando la versión 2.0 de la plataforma y planificando la migración del servicio actual a dicha versión.

1.2.5. Sistemas *SIU*

Los sistemas *SIU* que la *DGTIC de la Universidad Nacional de Entre Ríos* utiliza son:

- **Pilagá:** sistema web de gestión presupuestaria, financiera y contable.
- **Mapuche:** sistema de recursos humanos.
- **Guaraní:** sistema para la gestión académica.
- **Diaguita:** sistema de compras, contrataciones y patrimonio.
- **Kolla:** sistema de gestión de encuestas.
- **Wichi:** sistema de información gerencial.
- **Comdoc:** sistema web de seguimiento electrónico de documentación.

[4]http://moodle.org/

[5]Debian Etch (4.0) salió el 8 abril de 2007 y el fin del soporte del mismo fue a partir del 15 de febrero de 2010. Para más detalle, ver https://wiki.debian.org/DebianEtch

2. Capítulo II

2.1. Antecedentes

Este trabajo tiene como antecedentes varias experiencias similares que van desde el surgimiento de la filosofía de *DevOps*, con su metodología de trabajo y su concepción en la relación entre las áreas de desarrollo y operaciones, hasta su implementación en alguna empresa perteneciente al Estado Nacional, arrojando en todos los casos resultados positivos.

A continuación se tratarán algunos de los antecedentes que se consideran más destacados e influyentes para este trabajo.

2.1.1. Flickr

En el año 2009 John Allspaw y Paul Hammond dan una presentación titulada "10 deploys per day - Dev & Ops cooperation at Flickr"[19]. En dicha presentación, plantean la tensión que suele existir entre las áreas de desarrollo y de operaciones de cualquier organización y como esa relación genera dificultades en el negocio.

De manera interesante, indican que el objetivo del área de operaciones no es, como se cree normalmente, mantener el sitio estable y rápido, sino hacer posible el negocio, el cuál también resulta ser el objetivo del área de desarrollo. En este sentido, indican que si el negocio requiere que el sitio esté caído por dos horas así debe ser.

Con esta idea en mente y planteando un objetivo en común, ambos sugieren que es necesario dejar los estereotipos de lado y lograr que los administradores de sistemas piensen como desarrolladores y los desarrolladores piensen como administradores de sistemas. De esta manera, cada uno puede comprender las necesidades del otro y se puede generar la sinergia necesaria para alcanzar el despliegue continuo de cambios como forma de trabajo, lo que redundará en beneficios para el negocio.

Como dato interesante, es en esta presentación donde surge el término *DevOps*, cuya filosofía es la base que se toma como punto de partida para esta tesina.

2.1.2. PAMI

En la Puppetconf[6] del año 2014, Pablo Wright presentó la experiencia de haber implementado *Puppet*[7] en PAMI[8][43]. Explica cómo mejoraron el flujo de trabajo incluyendo ambientes de testing, de aseguramiento de calidad y de pre-producción y también la resistencia a trabajar con la metodología de *DevOps* que tuvieron de parte del personal, debido a que inicialmente demanda mucho trabajo, lo que lleva a pensar que hacer las cosas manualmente es más rápido.

Otro tema interesante que se menciona en esta exposición, es cómo se automatiza el manejo del firewall, un equipamiento dedicado de la empresa F5, interactuando con la API del equipo por medio de una librería de *Puppet*, mostrando que las herramientas de automatización de la infraestructura también son aplicables a equipamiento de red. En este sentido, incluso plantea como objetivo a futuro agregar más automatización en la red, incluyendo switches y routers.

2.1.3. Facebook

En una presentación dada durante una ChefConf[9] en el año 2013, Phil Dibowitz[26] cuenta cómo en Facebook lograron manejar una infraestructura de decenas de miles de servidores con un equipo de tan sólo cuatro personas. Además, en su presentación cuenta las características que considera que son importantes encontrar en un sistema de administración de configuraciones.

2.1.4. Automated Labs

Jeffrey Hulten explica en una charla titulada "Using Kanban and Chef"[32] su experiencia en el trabajo en grupo usando Chef[10] y cómo *Kanban* ayudó a su equipo a organizar de mejor manera el trabajo.

---

[6]Conferencia organizada por la empresa Puppet Labs.

[7]Ver la sección [B.2] en la página 93 para más información sobre Puppet.

[8]Formalmente llamado Instituto Nacional de Servicios Sociales para Jubilados y Pensionados, PAMI es una obra social estatal que brinda cobertura a jubilados y pensionados en todo el territorio nacional argentino.

[9]ChefConf es una conferencia organizada por Opscode, la empresa que desarrolla Chef.

[10]Ver la sección [B.3] en la página 95 para más información sobre Chef.

2.2. Bases teóricas

La investigación realizada en este trabajo, junto con las propuestas asociadas, tienen un marco teórico que se usa como base para el desarrollo del mismo. En este aspecto, la teoría alrededor de las metodologías ágiles constituyen la base más destacada, conjugando las diferentes metodologías conocidas, varias de las cuáles se detallan a continuación. Es importante considerar que, además, la corriente filosófica y metodológica de mayor influencia en este trabajo es la de *DevOps*, que se explica hacia el final de este capítulo.

2.2.1. Metodologías ágiles

La ley de Conway[24] dice que "cualquier organización que diseña sistemas producirá un diseño cuya estructura es una copia de la estructura de comunicación de la organización", y es debido a esto que se buscan mecanismos para poder mejorar la comunicación y la eficiencia de los equipos de trabajo, considerando principalmente entre estos mecanismos a las metodologías ágiles.

Dichas metodologías ágiles tienen su piedra fundamental en un manifiesto conocido como el manifiesto por el desarrollo ágil de software[10] que dice:

- **Individuos e interacciones** sobre procesos y herramientas.
- **Software funcionando** sobre documentación extensiva.
- **Colaboración con el cliente** sobre negociación contractual.
- **Respuesta ante el cambio** sobre seguir un plan.

De lo anterior se aclara que, mientras los ítems de la derecha son valorados, se consideran de mayor valor aún los respectivos de la izquierda. El mencionado manifiesto tiene, además, los siguientes doce principios:

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al período de tiempo más corto posible.

- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

2.2.2. Lean Software

En el desarrollo de software no hay una fórmula mágica, existen en cambio algunas teorías acerca de cómo aumentar la productividad, y son las áreas de manufactura y logística las que han desarrollado un conocimiento más profundo sobre cuál es la mejor forma de incrementar la productividad[37].

Una manera de evitar la baja productividad es tomar las decisiones correctas de diseño y evitar cambiarlas en el proceso de desarrollo. Este era el enfoque de *GM*. Sin embargo, *Toyota* y *Honda* descubrieron una forma diferente para evitar la baja productividad y dicho descubrimiento fue consecuencia de una decisión incorrecta de diseño. Entonces se formuló una nueva estrategia que consiste en no hacer decisiones irreversibles en primer lugar, demorar las decisiones de diseño lo más que se pueda y, cuando sea necesario tomarlas, hacerlas con toda la información disponible.

Resulta evidente que, al demorar lo máximo posible la toma de las decisiones más importantes de diseño, la información que se puede reunir acerca del problema a resolver es mucho más abundante que cuando se inicia el proyecto. Esta forma de concebir el diseño es muy similar a la filosofía de manufactura *Just-in-time*, en la que *Toyota* es pionera: no decidir qué fabricar hasta que un cliente haga un pedido; entonces hacerlo lo más rápido posible.

El retrasar las decisiones de diseño no es la única diferencia de los modelos de fabricación de *GM* y *Honda* durante los años 80. En *GM* se tendía a tomar las decisiones de diseño críticas en las altas esferas gerenciales, mientras que en *Honda* las decisiones de diseño de un nuevo motor la tomaban los ingenieros como consecuencia de discusiones detalladas. A su vez, *GM* desarrollaba productos de forma secuencial, mientras que *Honda* utilizaba procesos concurrentes, involucrando la fabricación, el testeo y el mantenimiento del auto en el diseño mismo del auto.

El modelo de fabricación utilizado por *Honda* y *Toyota* en los años 80, se denomina *Lean development*.

2.2.3. Waste

El objetivo de la producción *Lean* es entregar los productos a los clientes de la forma más rápida y la manera de hacer esto es encontrar y eliminar las pérdidas[37].

Shigeo Shingo quien co-desarrolló el sistema de producción de *Toyota* junto con Taiichi Ohno, identificó 7 tipos de desperdicios (*wastes*) que son fácilmente recordados utilizando el acrónimo *DOTWIMP*:

1. **Defects:** *Lean* se centra en prevenir defectos en lugar del tradicional "encontrar y arreglar".
2. **Overproduction:** producir más de lo que se necesita o producir antes de que se lo necesite.
3. **Transportation:** movimiento innecesario de las partes del proceso.
4. **Waiting:** personas o partes que esperan para el siguiente paso de producción.
5. **Inventory:** todo el material, trabajo en progreso y productos finalizados que no han sido procesados.

6. **Motion:** movimiento de personas o equipamiento más del que es necesario para hacer un procesamiento.

7. **Processing:** excesivo procesamiento por encima del estándar requerido por el cliente.

2.2.4. Kanban

En el año 1947 Taiichi Ohno, quien fuera vicepresidente de Toyota, impuso en la compañía la filosofía "Justo a Tiempo", que consiste en realizar "lo que se necesita, cuando se necesita, en la cantidad que se necesita" y que aún hoy regula el proceso de fabricación de la mencionada empresa[41]. A raíz de dicha filosofía se desarrolló el sistema *Kanban*, siendo este último un término Japonés que en Español podría traducirse como "tarjeta" o "tarjeta de señalización". Por medio de la aplicación del sistema *Kanban*, Toyota logró no sólo incrementar su productividad sino que también redujo el desperdicio, las inconsistencias y los requerimientos irracionales.

Si bien *Kanban* fue concebido inicialmente como una forma de optimizar los procesos de manufactura en un entorno industrial, los mismos principios fueron tomados luego y utilizados en otros ámbitos, como menciona David Anderson en su libro titulado *Kanban*[20].

El autor define que los principales objetivos de *Kanban* son:

- Limitar el *Work-in-Progress*.
- Visualizar el flujo de trabajo.
- Medir y optimizar dicho flujo.
- Hacer explícitas las políticas de procesos.
- Administrar cuantitativamente.

Kanban utiliza un sistema de tarjetas organizadas en columnas que permiten visualizar de forma simple y rápida el trabajo pendiente, el trabajo en curso y el trabajo realizado. Por la sencillez del método y su alta efectividad para mejorar la organización de un equipo de trabajo y gestionar las tareas, suele ser una de las metodologías ágiles más adoptadas en el campo del desarrollo de software. Existen múltiples plataformas que implementan *Kanban*,

siendo por ejemplo *Trello*[11] una de las más conocidas en la actualidad.

2.2.5. Teoría de las Restricciones

La teoría de las restricciones[12], desarrollada por Eliyahu Goldratt en la novela The Goal[45] indica que cualquier sistema tiene algún eslabón en toda su cadena de actividades que es el más débil de todos y el que impone la restricción al sistema. En este sentido, para mejorar el sistema completo se debe entonces reforzar dicho eslabón. Cuando se trabaje lo suficiente y ese eslabón deje de ser la restricción del sistema aparecerá entonces uno nuevo sobre el que habrá que trabajar. En esencia, la teoría de las restricciones se basa en cinco puntos:

- Identificar los eslabones que limitan al sistema.
- Determinar la manera de mejorar dichos eslabones.
- Orientar todos los esfuerzos a cumplir con la decisión anterior.
- Eliminar la restricción identificada o, en su defecto, disminuirla al máximo.
- Cumplido el paso anterior, volver al primer paso para identificar nuevas restricciones.

2.2.6. DevOps

Por ser relativamente nuevo y ciertamente amplio en algún sentido, pueden encontrarse múltiples interpretaciones alrededor del término *DevOps*. A continuación, se reunirán algunas de las diferentes definiciones que han sido de influencia para este trabajo, de manera de poder clarificar qué se entiende en el universo de TI por *DevOps* y cuál es la visión que esta tesina tiene acerca de *DevOps*.

DevOps se refiere al movimiento profesional emergente que propugna una relación laboral colaborativa entre las áreas de desarrollo y de administración de sistemas enuncia Gene Kim[34], quién además sostiene que, como

[11] https://trello.com

[12] También conocida como Teoría de las Limitaciones.

consecuencia de *DevOps*, se obtiene un flujo rápido de trabajo planificado, mientras que, de manera simultánea, se incrementa la confiabilidad, estabilidad y seguridad de los ambientes productivos.

Según el mismo autor, los orígenes de *DevOps* se pueden situar temporalmente en el año 2009, ante la convergencia de numerosos movimientos:

- Las conferencias Velocity, en particular la presentación "10 deploys per day - Dev & Ops cooperation at Flickr"[19], mencionada anteriormente en la sección de antecedentes de este trabajo[13].
- El movimiento de *infrastructure as code* liderado por Mark Burgess y Luke Kanies, el movimiento *Agile infrastructure* promovido por Andrew Shafer y el movimiento *Agile system administration*, representado por Patrick DeBois.
- El movimiento de *Lean Startup*, promovido por Eric Ries.
- El movimiento de integración y lanzamiento continuo de Jez Humble.
- La global disponibilidad de las tecnologías de cloud y de plataforma como servicio[14].

Profundizando un poco en los orígenes y el desarrollo de *DevOps*, Damon Edwards dedica un post titulado "The (Short) History of DevOps"[29] donde hace un repaso de la historia de *DevOps* en un video de poco menos de doce minutos de duración.

Por su parte, Gene Kim explica que la relación entre *DevOps* y el desarrollo ágil es complementaria y que *DevOps* extiende y completa el proceso de integración y lanzamiento continuo, asegurándose que el código está listo para producción y que agregará valor para los clientes.

Otra persona que acerca una definición de *DevOps* es Adam Jacob, quién brinda una discusión alrededor de qué es *DevOps* y qué no, e introduce lo que él denomina *DevOps Kung-fu*[33], entendiendo por Kung-fu no el arte marcial sino el significado detrás, que es "la excelencia alcanzada a través de largas prácticas de las habilidades de uno", según el mismo Adam explica. La

[13]Ver la sección [2.1.1] en la página 16 para más información sobre la presentación "10 deploys per day - Dev & Ops cooperation at Flickr".

[14]PaaS, Platform as a Service.

filosofía de *DevOps Kung-fu* es construida por quiénes la practican, comenta Adam, y es abierta a todos. Tal es así que existe un repositorio en *Github* con una presentación sobre *DevOps Kung-fu*[46] al que cualquier persona puede contribuir.

Hasta aquí se ha tratado a *DevOps* como un movimiento o una filosofía, enfocado en la relación cooperativa entre las áreas de desarrollo y administración de sistemas. Existen otros enfoques, como el dado por Aliza Earnshaw en un post titulado "What is a DevOps Engineer?"[28], que entiende a *DevOps* como el rol de un profesional de la informática que, como comenta la mencionada autora, si bien no se llega a ser un *DevOps* por medio de una carrera formal, se transforma en *DevOps* un desarrollador que se interesa en el despliegue de aplicaciones y las operaciones de red, o un administrador de sistemas que, apasionado por escribir código, mueve su foco de trabajo hacia el desarrollo, desde donde busca mejorar la planificación de las pruebas y el despliegue de aplicaciones.

En su intención de caracterizar a un *DevOps*, la autora enuncia una serie de atributos básicos que un *DevOps* reúne:

- Habilidad de utilizar una variedad de tecnologías y herramientas de código abierto.
- Habilidad de escribir *scripts* y desarrollar.
- Experiencia con sistemas y operaciones de TI.
- Comodidad en la prueba y despliegue frecuente e incremental de código.
- Gran comprensión de herramientas de automatización.
- Habilidad para la administración de información.
- Fuerte focalización en los resultados del negocio.
- Capacidad de comunicación abierta, de colaboración y de atravesar las fronteras funcionales.

Es importante destacar que la idea de *DevOps* como un rol profesional que brinda Aliza se opone a lo expresado por los autores citados antes, pero resulta de interés diferenciar de alguna manera al desarrollador y/o administrador de sistemas que trabaja siguiendo la filosofía *DevOps* del que no lo hace. Por ello, en el contexto de este trabajo, se entiende como *DevOps* tanto a la manera de relacionarse de forma cooperativa entre las áreas de desarrollo

y de administración de sistemas, a la metodología de trabajo planteada en base a las pruebas y el despliegue continuo de código y al rol que cumple un profesional de la informática al llevar adelante esta filosofía y metodología definida, caracterizado por los atributos que Aliza Earnshaw enuncia.

Relacionando la idea de *DevOps* con la teoría de las restricciones explicada anteriormente, se puede entender que *DevOps* surge como una forma de atacar precisamente el eslabón más débil en un sistema de desarrollo y despliegue de aplicaciones, junto sus pruebas y puesta en producción, en el cuál las metodologías ágiles implementadas en los desarrollos de software han llevado a que el área de operaciones de cualquier organización se transforme en el cuello de botella del sistema y que, por lo tanto, requiera ser mejorada.

2.2.7. Sistemas Inmutables

El concepto de sistemas inmutables hace referencia a un sistema que nunca cambia, lo que significa que ese sistema será de sólo lectura. Tal es la idea detrás de *Docker*[15] y será el punto de partida para las implementaciones llevadas adelante en este trabajo con la mencionada herramienta. Al mismo tiempo, un sistema inmutable es un sistema descartable, es decir que puede destruirse y volver a crearse sin que eso suponga ningún inconveniente[23].

De esta manera, los sistemas inmutables deben poder crearse para ejecutar un servicio y destruirse cuando ya no son necesarios o cuando están obsoletos, reemplazándose por otro sistema inmutable actualizado. Asociado a esto, el tiempo de vida de un sistema inmutable es el tiempo durante el cuál se lo necesita, que puede ir desde unos pocos segundos para, por ejemplo, ejecutar un comando, hasta meses si el mismo está prestando un servicio que requiere un uso más prolongado. De todas maneras, el tiempo de vida de los sistemas de este tipo difícilmente se extienda más allá de algunos pocos meses.

En una publicación en su blog[25], Michael DeHaan[16] explica por qué él considera que la infraestructura inmutable es el futuro y enuncia una reflexión sumamente interesante en la que expresa que la única forma de tener la certeza de que un sistema no haya sufrido cambios que uno no espera es destruirlo por completo. Así, continúa DeHaan, poder asegurarse que uno no

[15]Ver la sección [A.3.4] en la página 86 para más información sobre Docker.

[16]Michael DeHaan es el creador de Ansible, un sistema de automatización que se analizará y utilizará más adelante en este mismo trabajo.

está atado a un sistema y que puede reconstruirlo en caso de una emergencia implica precisamente perder el miedo a destruirlo y volver a crearlo.

Un sistema inmutable, insistiendo en el concepto, no acepta entonces ningún tipo de modificación. Cualquier cambio, por mínimo que sea, exige destruir el sistema y crear uno nuevo con el cambio incorporado. Esto que puede parecer extremista constituye en realidad uno de los mayores atractivos de este tipo de sistemas debido a que garantiza, por ejemplo, que se pueda trabajar en cualquier ambiente con **exactamente** la misma versión del sistema productivo, y se destaca la palabra exactamente porque es justamente lo que los sistemas inmutables aseguran.

3. Capítulo III

3.1. Definiciones

3.1.1. Ambientes

Se entiende por ambiente de:

- **Desarrollo**: el ambiente de desarrollo es en el cuál los desarrolladores construyen el software.
- **Testing**: es el ambiente donde se publica el software en fase de pruebas para que sea probado por un grupo definido de personas, entre las que se incluye el usuario final o representantes del mismo.
- **Pre-producción**: es la instancia previa a producción, y consiste en un ambiente replicado e idéntico al productivo. En este entorno se verifica el correcto funcionamiento de la aplicación y se realizan los ajustes necesarios en caso de no ser así, evitando que los problemas se descubran en el pasaje a producción.
- **Producción**: es el ambiente que tiene todos los servicios productivos. Este ambiente cuenta con políticas estrictas en cuanto al acceso y la administración del mismo.

3.2. Herramientas a utilizar

3.2.1. Gestión de proyectos

Al comenzar la implementación de este trabajo se evaluaron algunas herramientas para gestionar los proyectos de la *DGTIC de la Universidad Nacional de Entre Ríos* y se seleccionó la herramienta *Kanboard*[17], que tiene las siguientes características:

- Utiliza la metodología *Kanban*.
- Se visualiza claramente el trabajo sobre un tablero.

[17] http://kanboard.net/

- Maneja múltiples proyectos y usuarios.
- Permite realizar reportes y análisis.
- Es accesible por medio de un navegador.
- Es de código abierto.

En el contexto de este trabajo, se instaló la herramienta para uso interno en la *DGTIC de la Universidad Nacional de Entre Ríos*, de manera de mejorar la organización de las actividades y los proyectos.

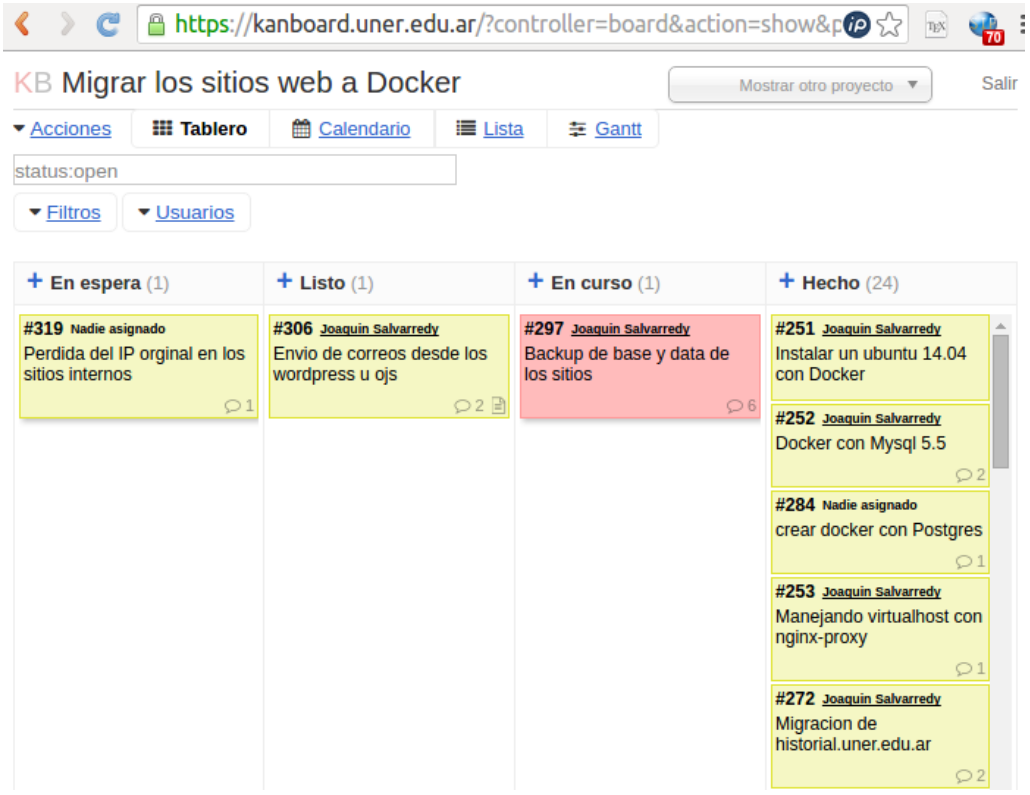

Figura 1: Gestionando proyectos con la metodología *Kanban*

3.2.2. Herramienta de automatización

Posterior al análisis de las diferentes herramientas de aprovisionamiento y automatización, cuyas características generales pueden encontrarse en los

anexos de este trabajo, se decidió utilizar *Ansible*[18] por los motivos que se explican a continuación.

- La curva de aprendizaje de *Ansible* es mucho menor que la del resto de las herramientas[35], principalmente si se la compara con herramientas como *Chef* y *Puppet*.
- Utiliza tecnologías bien conocidas en el área de operaciones y las soluciones se escriben por medio de sencillos archivos de texto.
- No requiere la instalación de ningún agente en los servidores remotos.
- Aplica correctamente al caso de estudio.

3.3. Gestión de aplicaciones

La gestión de las aplicaciones que brinda la *DGTIC de la Universidad Nacional de Entre Ríos* tiene algunas dificultades que son comunes en muchas organizaciones y constituye uno de los puntos a resolver. Como parte de este trabajo, y con la intención de poder ordenar la administración de los sistemas brindados, se diseñaron una serie de procesos que intentan resolver algunas de las problemáticas más habituales y que mayor impacto tienen en la operatoria diaria y en el mantenimiento de la infraestructura. Dichos procesos serán tratados a continuación.

3.3.1. Proceso de solicitud de nueva aplicación

El problema en la gestión de las aplicaciones surge desde el momento mismo en que la *DGTIC de la Universidad Nacional de Entre Ríos* recibe una solicitud para desplegar un sistema. Esto ocurre porque, al no existir un proceso formal para el alta de las aplicaciones, tampoco se cuenta luego con la información necesaria para gestionarlas en el día a día. Dicha información implica mínimamente conocer quién es el responsable de la aplicación, los medios para contactarlo, la transmisión de las políticas de la prestación del servicio a los solicitantes del mismo y la aceptación por parte de estos de dichas políticas. Considerando estas y otras cuestiones es que se diseñó el siguiente proceso para realizar la solicitud de una nueva aplicación.

[18] Ver la sección [B.1] en la página 91 para más información sobre Ansible.

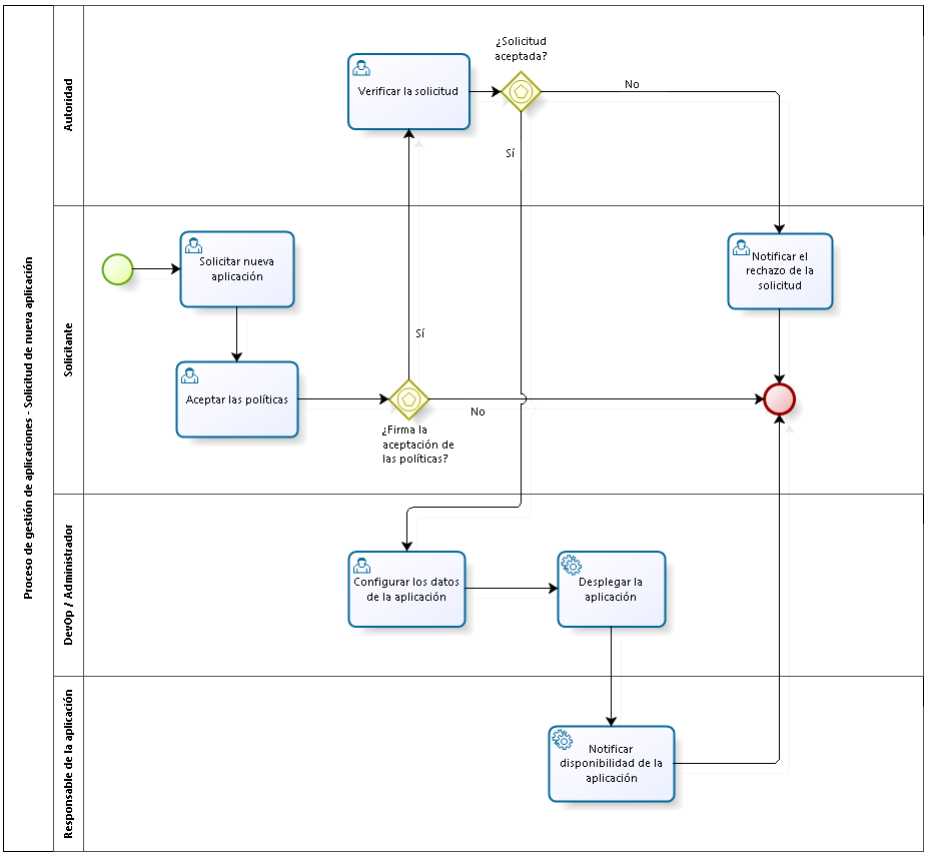

Figura 2: Solicitud de nueva aplicación

En el proceso anterior se presentan las siguientes tareas:

- **Solicitar nueva aplicación:** en este primer paso, con el que el solicitante de la aplicación inicia el proceso, se debe completar información relativa a la aplicación solicitada. Esto puede implicar aportar diferentes datos, dependiendo del tipo de aplicación solicitada y del ambiente en que se desee montar. Más allá de las particularidades de cada solicitud, lo que debe incluir de forma obligatoria es nombre y apellido del responsable de la aplicación junto con los datos de contacto, que mínimamente deben incluir un correo electrónico y un número de teléfono. Se debe tener en cuenta que, en muchos casos, el responsable de la aplicación puede no ser la misma persona que la solicita.
- **Aceptar las políticas:** la prestación del servicio por parte de la *DGTIC de la Universidad Nacional de Entre Ríos* debe definir ciertas políticas que tendrán que ser aceptadas por el solicitante para que

su aplicación pueda ser desplegada en la infraestructura de la mencionada Dirección. La aceptación de las políticas es entonces una condición excluyente para continuar con el proceso de solicitud de la aplicación. Más adelante, en este mismo trabajo, podrá verse algún delineamiento inicial de lo que podrían definir las políticas para el caso de la puesta en producción de las aplicaciones en testing[19].

- **Verificar la solicitud:** toda solicitud recibida debe ser aprobada por alguna autoridad de la *Universidad Nacional de Entre Ríos* para poder darle curso a la misma. Nuevamente, la no aprobación de la solicitud por parte de una autoridad impide que se pueda continuar con el proceso.
- **Notificar el rechazo de la solicitud:** en el caso de que la autoridad correspondiente hubiera rechazado la solicitud, se deberá notificar al solicitante de esta situación.
- **Configurar los datos de la aplicación:** si la solicitud fue aprobada, entonces el administrador de sistemas debe realizar todas las configuraciones preliminares para poder desplegar la aplicación.
- **Desplegar la aplicación:** la aplicación se despliega efectivamente en la infraestructura de la *DGTIC de la Universidad Nacional de Entre Ríos* según los datos configurados por el administrador de sistemas en el paso anterior. Esta tarea puede implicar múltiples actividades que no se detallan en este proceso porque el foco está puesto en la solicitud.
- **Notificar disponibilidad de la aplicación:** cuando la aplicación ya ha sido desplegada, se notifica directamente al contacto definido como responsable de la misma. Esta tarea es automatizada y la realiza la misma herramienta de gestión, debido a que ya cuenta con los datos de contacto de dicha persona.

3.3.2. Reempadronamiento de aplicaciones

En el punto anterior se definió un proceso para la solicitud de nuevas aplicaciones. Es importante notar que la *DGTIC de la Universidad Nacional de Entre Ríos* ya cuenta además con muchas aplicaciones desplegadas en

[19]Si bien el ambiente de testing es un ambiente de pruebas, puede considerarse que en él hay servicios que, aunque tienen características particulares por las cuáles se instalan en un ambiente específico, los servicios deben permanecer activos y estables.

su infraestructura. En este caso, se debe relevar la información necesaria para poder contar con los datos requeridos, como si las aplicaciones hubieran pasado por el proceso definido en el apartado anterior. Para ello, de cada aplicación se deberá:

- Obtener un contacto que pueda completar la planilla de solicitud, que será considerada a partir del momento en que se contacte a dicha persona. Esta planilla de solicitud sigue los mismos lineamientos explicados en la tarea "Solicitar nueva aplicación" del proceso anterior.
- Solicitar la firma confirmando aceptación de las políticas por parte del responsable o alguna autoridad.

3.3.3. Gestión de aplicaciones en testing

Entre las tareas de mantenimiento, una que suele ser foco de grandes dificultades en la gestión es la del manejo de la caducidad de las aplicaciones desplegadas en testing. El motivo es que el ambiente de testing es, precisamente, una instancia en la que se realizan pruebas sobre las aplicaciones, que pueden ser pruebas de nuevas funcionalidades, pruebas de aplicaciones de terceros, pruebas de concepto o cualquier otra cuestión que requiera verificar el funcionamiento de un sistema.

Estas aplicaciones son desplegadas por la *DGTIC de la Universidad Nacional de Entre Ríos*, pero en muchos casos surgen como consecuencia de pedidos de terceros. El problema que se da en esta instancia para la *DGTIC de la Universidad Nacional de Entre Ríos* es que resulta difícil determinar cuáles de todas las aplicaciones desplegadas en su infraestructura de testing están realmente siendo utilizadas y cuáles ya han dejado de ser necesarias.

Se plantea entonces, a continuación, el ciclo de vida de las aplicaciones en testing y se presenta luego un proceso para gestionar la caducidad de las mismas.

3.3.3.1 Ciclo de vida

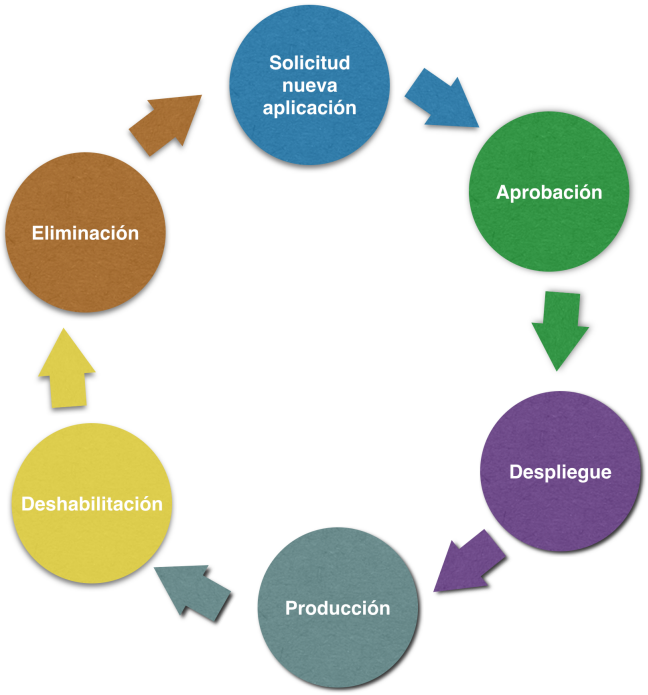

Figura 3: Ciclo de vida de una aplicación en testing

3.3.3.2 Proceso de gestión de la caducidad

Para resolver la problemática planteada respecto de la gestión de las aplicaciones en testing, se diseñó un proceso que maneja su caducidad de forma totalmente automatizada. Este proceso garantiza, en todo momento, la vigencia de las aplicaciones desplegadas en testing e independiza al personal de la *DGTIC de la Universidad Nacional de Entre Ríos* de la tarea de deshabilitar y/o borrar de su infraestructura aquellos sistemas que ya no son necesarios. En el siguiente gráfico se puede apreciar el proceso diseñado, cuyas tareas se explican a continuación del mismo.

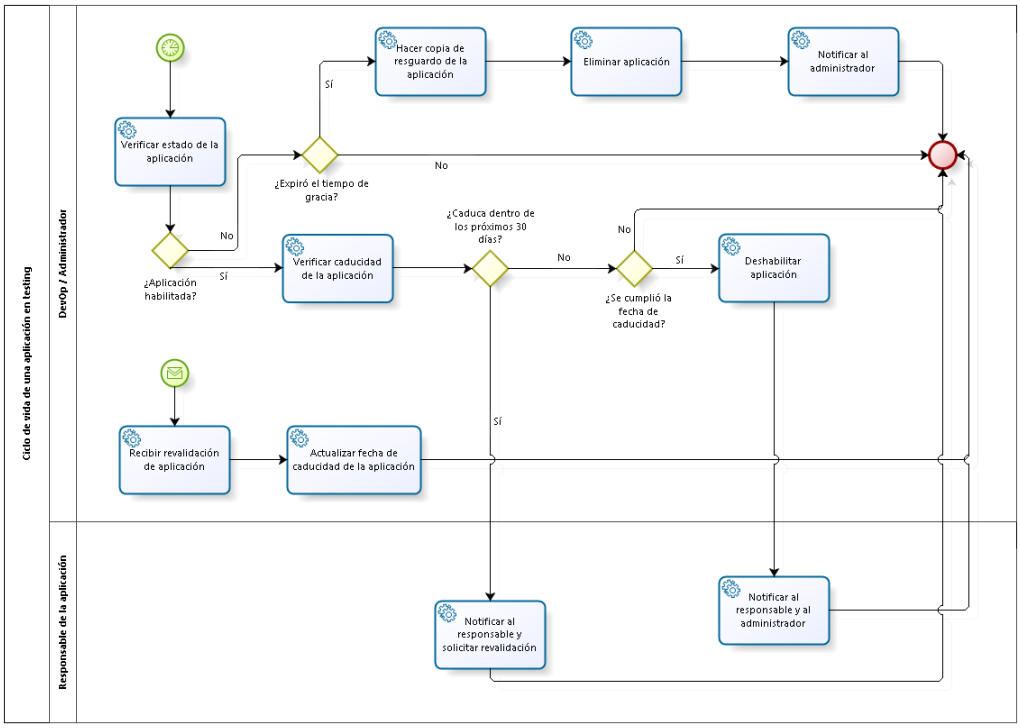

Figura 4: Proceso de gestión de aplicación en testing

- **Inicio del proceso:** el proceso puede iniciarse por dos eventos diferentes. Por un lado, por una tarea programada y periódica que tendrá como objetivo verificar la caducidad de las aplicaciones y realizar acciones en consecuencia según cuál sea el resultado de su evaluación; por el otro, a través de una solicitud de revalidación de una aplicación recibida por un usuario.
- **Verificar estado de la aplicación:** esta tarea toma una aplicación y verifica si la misma se encuentra habilitada (activa) o no.
- **Verificar caducidad de la aplicación:** en caso de que la aplicación esté activa, debe verificarse la fecha de caducidad asociada a la misma. En este sentido, si la fecha de caducidad está dentro de los siguientes treinta días se debe comenzar el proceso de revalidación para verificar si la aplicación aún es necesaria. Si la fecha de caducidad no estuviera dentro de los siguientes treinta días podría ser por dos motivos: por un lado, porque la fecha de caducidad ya se haya cumplido (en cuyo caso tendrá una fecha anterior a la actual) o porque falten más de treinta días para tal fecha. Si este último es el caso entonces el proceso puede terminar en este punto directamente.

- **Notificar al responsable y solicitar revalidación:** cuando la fecha de caducidad de una aplicación está dentro de los treinta días hábiles siguientes al inicio del proceso, se debe solicitar al responsable de la aplicación que haga la correspondiente revalidación de la misma, de manera de saber si la aplicación debe continuar activa o si puede darse de baja. Una vez enviada la notificación al responsable el proceso ya no tiene otra tarea que ejecutar por lo que puede terminar.

- **Deshabilitar aplicación:** en caso de que la fecha de caducidad se hubiera cumplido, la aplicación debe ser deshabilitada. El efecto de esta tarea es que la aplicación deje de estar en línea.

- **Notificar al responsable y al administrador:** cuando se deshabilita una aplicación, se notifica al administrador y al responsable de la aplicación.

- **Hacer una copia de resguardo de la aplicación:** si el tiempo de gracia[20] de la aplicación expiró, la misma será borrada. Por ello, previamente se hace una copia de seguridad para poder restaurar luego la aplicación en caso de ser necesario.

- **Eliminar aplicación:** en esta instancia, se elimina la aplicación junto con todos sus datos y las configuraciones de los servicios que pudieran existir aún.

- **Notificar al administrador:** al eliminar una aplicación, se genera un aviso al administrador.

- **Recibir revalidación de aplicación:** si el proceso se inicia a través de un pedido de revalidación, el sistema recibe y procesa dicho pedido.

- **Actualizar fecha de caducidad de la aplicación:** con el pedido de revalidación aceptado, se actualiza la fecha de caducidad de la aplicación, renovando su vigencia por un tiempo determinado por el usuario, el administrador o las políticas definidas.

---

[20]Se entiende por tiempo de gracia al período de tiempo que puede transcurrir desde el momento en que la aplicación se deshabilitó hasta que la misma sea eliminada por completo.

3.3.4. Proceso de gestión de cambios en producción

En el proceso de gestión de cambios en las aplicaciones en producción se buscó atacar varios problemas detectados:

- Los desarrolladores tienen acceso como administrador a los servidores, pudiendo realizar ellos mismos cambios en producción, no sólo de las aplicaciones sino también de los servicios.
- La gestión de los cambios es manual, perdiendo el registro de cuáles han sido los cambios.
- No existe un ambiente de pre-producción que permita evaluar, en un entorno idéntico al productivo, el impacto de los cambios aplicados.
- Los administradores de sistemas pierden el control de los propios servidores como consecuencia del acceso irrestricto de los desarrolladores a los servidores.

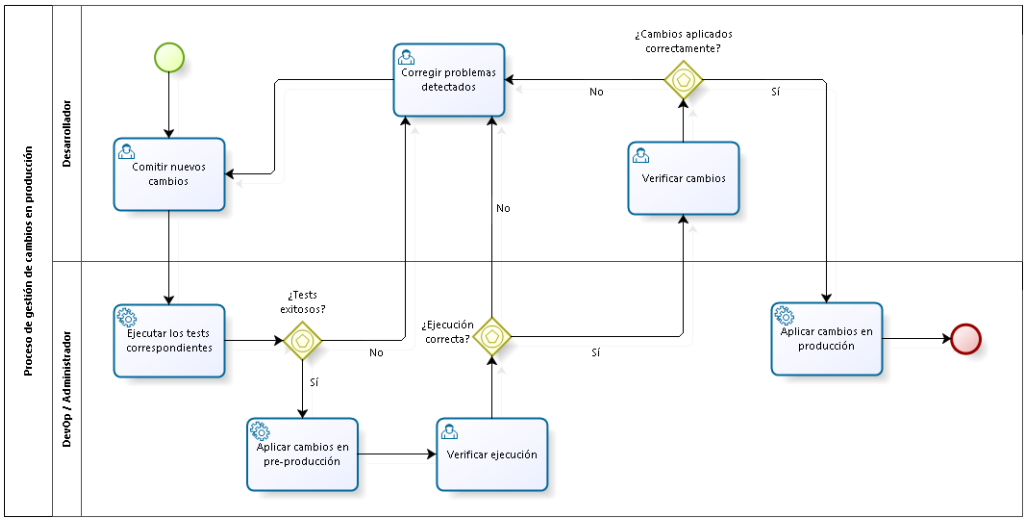

Figura 5: Gestión de cambios en aplicaciones en producción

A continuación se detallan cada una de las tareas del proceso.

- **Comitir nuevos cambios:** esta tarea, ejecutada por un desarrollador, implica que se apliquen los cambios en el repositorio de GIT de la aplicación.

- **Ejecutar los tests correspondientes:** cada aplicación debería tener sus propios tests de código. En caso de ser así, luego de aplicar y subir los cambios al repositorio de GIT, los mismos son enviados a un servidor de integración continua que deberá ejecutar todos los tests contra el código subido. Si el código pasa con éxito todos los tests, entonces el mismo servidor de integración continua dispara un proceso que aplicará los cambios en la infraestructura de pre-producción.
- **Aplicar cambios en pre-producción:** en esta tarea se toma la versión del código testeada en el paso anterior y se despliega en pre-producción.
- **Verificar ejecución:** es posible que el despliegue en pre-producción falle. Por ello es necesario que el administrador verifique el resultado del despliegue de la aplicación y notifique al desarrollador del problema o le pida, en su lugar, que chequee los cambios aplicados.
- **Corregir problemas detectados:** en cualquiera de los casos que se detecte un inconveniente será el desarrollador el encargado de verificar el problema, buscar la solución correspondiente y volver a intentar el despliegue del código con los nuevos cambios incorporados. Es importante notar que, en este punto, puede requerir la colaboración del *DevOps*, debido a que los problemas pueden tener diferentes motivos, como puede ser la falta de alguna librería o paquete en el servidor.
- **Verificar cambios:** en el caso de que el despliegue del nuevo código se haya concretado con éxito, el desarrollador deberá verificar en pre-producción que los cambios se hayan reflejado y que el sistema funcione correctamente.
- **Aplicar cambios en producción:** cumplida la tarea anterior, el administrador ejecutará en producción la misma tarea que se ejecutó de forma automática en pre-producción y que dará como resultado la aplicación desplegada de forma correcta y funcionando.

3.4. Aulas Virtuales

La problemática de las aulas virtuales se presentó en el primer capítulo de esta tesina[21]. En el contexto de este trabajo, se buscó solucionar dicho aspecto

[21] Ver la sección [1.2.4] en la página 15 para más información sobre las aulas virtuales.

de la infraestructura de servicios de la *DGTIC de la Universidad Nacional de Entre Ríos* y, para ello, se generó un rediseño de la arquitectura seguido de la implementación de la misma utilizando herramientas de automatización para los servidores web. Finalmente, se ejecutó la migración del servicio a la nueva arquitectura.

A continuación se presenta el mencionado rediseño, repasando las herramientas utilizadas e incluyendo la instalación, en paralelo, de la versión 2.0 de *Moodle*.

3.4.1. Diseño de arquitectura

Para el diseño se tuvieron en cuenta varios factores[44], como la segmentación de los servicios, la facilidad de reinstalación y los mecanismos de backups. En el siguiente gráfico se muestra el diseño simplificado de la arquitectura.

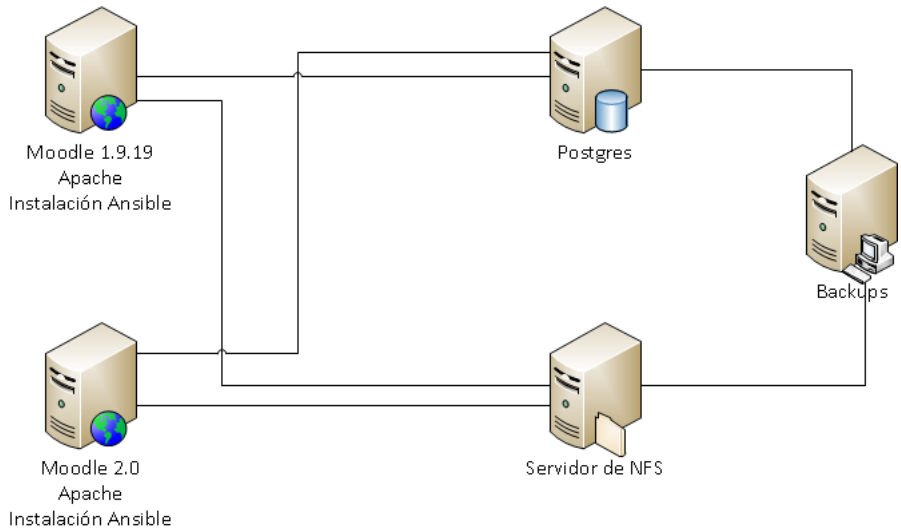

Figura 6: Arquitectura de la Infraestructura del Moodle

En la arquitectura anterior se puede ver que se realizó una separación entre los servidores web, el almacenamiento de los datos dinámicos de las plataformas *Moodle* y el servicio de base de datos. Además, se incorporó equipamiento para realizar los backups de las aulas virtuales. Con el diseño propuesto se buscó:

- Separar los datos dinámicos, que deben ser resguardados y persistidos[2], de la configuración de los servidores web.

- Aprovisionar los servidores web de manera automática usando herramientas de orquestación.
- Independizar el servicio de base de datos del servicio web, dedicando para ello un *cluster* de servidores a tal fin.
- Utilizar la red de datos para la comunicación entre los diferentes servidores.

3.4.2. Implementación

En la implementación de la arquitectura presentada se utilizó una serie de herramientas, cada una con fines específicos. Las mismas se listan a continuación[22].

- *Ansible*.
- *Gitlab*.
- *Packer*.
- *Vagrant*
- *Virtualbox*.

Para desarrollar y probar los *Playbooks*[23] de *Ansible* se utilizó *Vagrant* en combinación con *Virtualbox*. Esto permitió probar paso a paso sobre una arquitectura local cada requerimiento. En todo momento se utilizó GIT y *Gitlab* para versionar el código que se fue escribiendo, de manera de poder mantener una historia en el desarrollo.

Así, se logró definir el *Playbook* para los servidores web que se muestra a continuación. En él se pueden ver comentarios que explican paso a paso qué hace el *Playbook*[24].

---

[22]Remitirse a los anexos y el glosario para más detalle sobre las mismas.

[23]Ver la sección [B.1.1] en la página 91 para más información sobre Playbooks en Ansible.

[24]Las tildes fueron intencionalmente omitidas en el siguiente ejemplo para evitar problemas de codificación en diferentes sistemas.

```
  # La siguiente linea define un grupo de hosts sobre los que
  # se aplicara el playbook.
- hosts: moodle191
  # Lo siguiente incorpora lo definido en el archivo
  # yaegashi.blockinfile a este playbook. Puede verse
  # de manera similar al include en un lenguaje de
  # programacion.
  roles:
    - yaegashi.blockinfile
  vars:
      sourceslist: sources.list-uner-wheezy
  tasks:

      # Copia el archivo ./data/sources.list-uner-wheezy en la
      # maquina local a la ubicacion /etc/apt/sources.list del
      # servidor aprovisionado.
    - name: Definiendo el mirror para descargar paquetes
      copy: src=data/{{sourceslist}} dest=/etc/apt/sources.list

    - name: Actualizando los repositorios del sistema
      shell: apt-get update

    - name: Instalando algunos paquetes necesarios
      action: apt pkg={{item}} state=present
      with_items:
       - rsync
       - screen
       - ntpdate
       - sudo
       - resolvconf
       - nfs-common
       - clamav
       - unzip
       - aspell-es
       - zip
       - locales
       - atsar
       - texlive-latex-base
       - imagemagick

    - name: Instalando Apache, PHP y otras dependencias
      action: apt pkg={{item}} state=installed
      with_items:
```

```
44          - apache2
45          - libapache2-mod-php5
46          - php5
47          - php5-pgsql
48          - php5-xmlrpc
49          - php5-cgi
50          - php5-gd
51          - php5-cli
52          - php5-curl
53          - php-apc
54
55        # Copia el archivo ./data/apache.conf en la maquina local
56        # a la ubicacion /etc/apache2/sites-available/default del
57        # servidor aprovisionado. Esto define el host virtual
58        # correspondiente.
59      - name: Copiando la configuracion por defecto de Apache
60        copy: src=data/apache.conf \
61              dest=/etc/apache2/sites-available/default
62
63        # Edita el archivo /etc/apache2/apache2.conf del servidor
64        # remoto para agregar el numero 30 a la linea que comienza
65        # con la palabra Timeout.
66      - name: Optimizando parametros de Apache
67        lineinfile: dest=/etc/apache2/apache2.conf regexp="^Timeout" \
68                    insertafter="^Timeout" line="Timeout 30"
69
70        # Agrega al archivo /etc/fstab una linea para especificarle
71        # montar desde el servidor con la IP 10.X.X.X el directorio
72        # /data/moodle191 en su propio directorio /data usando NFS.
73      - name: Agregando en fstab las entradas para que monte por NFS
74        blockinfile: |
75          dest=/etc/fstab
76          content='10.X.X.X:/data/moodle191   /data   nfs  defaults'
77
78        # Crea el directorio /data y le asigna propietario, grupo y
79        # los permisos correspondientes.
80      - name: Creando directorio data para montarlo por NFS
81        file: path=/data state=directory owner=www-data \
82              group=www-data mode=0775
83
84        # Agrega al CRON del equipo una linea para que ejecute cada
85        # 10 minutos el script /data/usr-share/admin/cron.php.
86      - name: Agregando la entrada al cron para que envie  mails
```

```
87        cron: name="moodle mail" minute="*/10" job="/usr/bin/php5 \
88             /data/usr-share/admin/cron.php > /dev/null"
89
90        # Instala un servidor de correo local en el servidor.
91      - name: Instalando el servidor de correo
92        action: apt pkg={{item}} state=present
93        with_items:
94         - postfix
95         - mailutils
96
97        # Define la configuracion IP del servidor copiando el
98        # archivo data/interfaces en la maquina local al archivo
99        # /etc/network/interfaces en la maquina remota.
100       # Las lineas debajo realizan una tarea similar copiando
101       # archivos de configuracion.
102     - name: Se copia la configuracion IP
103       copy: src=data/interfaces dest=/etc/network/interfaces
104
105     - name: Se copia la configuracion mailname
106       copy: src=data/mailname dest=/etc/mailname
107
108     - name: Se copia la configuracion del postfix
109       copy: src=data/main.cf dest=/etc/postfix/main.cf
110
111     - name: Se copia el firewall
112       copy: src=data/firewall.sh dest=/etc/init.d/firewall.sh \
113             owner=root group=root mode=0755
114
115     - name: Se copia el php.ini
116       copy: src=data/php.ini dest=/etc/php5/apache2/php.ini \
117             owner=root group=root
118
119     - name: Se copia el etc hosts
120       copy: src=data/hosts dest=/etc/hosts owner=root group=root
121
122     - name: Se copia la generaciones de locales
123       copy: src=data/locales.gen dest=/etc/locales.gen \
124             owner=root group=root
125
126     - name: Se copia el locale por deafult
127       copy: src=data/locale dest=/etc/default/locale \
128             owner=root group=root
129
```

```
130     - name: regenerar los locales
131       command: locale-gen -a
132
133     - name: Agregar el modulo rewrite al apache
134       command: a2enmod rewrite
135
136       # Agrega el firewall para que arranque al inicio.
137     - name: start firewall service
138       command: update-rc.d firewall.sh defaults
139
140       # Como ultimo paso, reinicia el servidor.
141     - name: Se reinicia el servidor para tomar los cambios
142       command: /sbin/reboot
```

La infraestructura de la *DGTIC de la Universidad Nacional de Entre Ríos* es gestionada utilizando *VMware* como plataforma de virtualización. Debido a que la mencionada herramienta cuenta con la posibilidad de definir *templates*[25][42] que luego se pueden utilizar para crear los servidores virtuales de forma más ágil, se adoptó dicha estrategia para reducir aún más los tiempos de despliegue de los servicios.

Con el *Playbook* anterior terminado[26] y definidos templates de diferentes sistemas operativos se realiza la instalación del servidor web con *Ansible*. Se opta finalmente por la versión 7.x del sistema operativo *Debian*.

3.4.3. Moodle 2.0

Siguiendo los lineamientos del diseño e implementación vistos hasta aquí, se procedió a realizar una instalación nueva del sistema *Moodle 2.x*. Para ello, se siguieron los siguientes pasos:

1. Se creó una nueva base de datos en el mismo servidor visto en la arquitectura planteada.

2. De igual manera, se utilizó el mismo servidor de archivos.

[25]Su traducción al español es plantilla, pero se mantiene el término en inglés por ser el de uso común.

[26]Se probó usando *Vagrant* y se verificó su correcto funcionamiento sobre *Debian*, en las versiones 6.x y 7.x de dicho sistema operativo

3. Se aplicó el *Playbook* definido anteriormente con pequeñas diferencias relacionadas a los nombres e IPs.

Con la plataforma instalada, se inicia un período de pruebas para analizar la estrategia de migración de datos desde la versión anterior y para verificar el funcionamiento global. Durante este período de pruebas fue necesario modificar algunos parámetros de configuración de la plataforma. Esta tarea se hizo, primero escribiendo los cambios necesarios en el *Playbook* correspondiente, cambios que luego se versionaron y finalmente se aplicaron. De esta manera, se puede garantizar que el código en producción es el mismo que se tiene disponible para replicar el servicio en una instancia futura.

Como ventaja adicional, al tener el código del *Playbook* versionado se cuenta con la historia de todos los cambios que se aplicaron en la infraestructura, al mismo tiempo que, por ser los *Playbooks* de lectura simple, se tiene una descripción precisa de la infraestructura, lo que sirve también como documentación.

3.5. Sistemas *SIU*

Los sistemas que provee el *SIU* tienen las mismas características comunes debido a que todos están desarrollados haciendo uso del *framework SIU-Toba*[27] el cual utiliza *PHP* como lenguaje de programación[28]. Por tal motivo, todos los sistemas tienen requerimientos de software muy similares.

Debe considerarse, además, que de varias de las aplicaciones de *SIU* existen múltiples instancias, tal es el caso de *Pilagá* que, con el fin de cada año de ejercicio, debe cerrarse exactamente con los datos que tenga al momento de dicho fin. De esta manera, pueden encontrarse las versiones *Pilaga 2011*, *Pilaga 2012*, *Pilaga 2013*, *Pilaga 2014* y *Pilaga 2015*, cada una con diferente código y una base de datos separada, aunque en todos los casos se utiliza *PostgreSQL* como motor de base de datos.

Se identifican las siguientes cuestiones que deben considerarse:

---

[27]http://www.siu.edu.ar/siu-toba/

[28]La *DGTIC de la Universidad Nacional de Entre Ríos* utiliza también el sistema *Comdoc* que fue desarrollado en realidad por el Ministerio de Economía y Finanzas Públicas de la Nación y cedido al *SIU* para su distribución en universidades nacionales. Este sistema, que utiliza *J2EE*, es el único de los sistemas de *SIU* que provee la *DGTIC de la Universidad Nacional de Entre Ríos* que no está desarrollado con el *framework SIU-Toba*.

- Migración de cada sistema a la nueva infraestructura.
- Instalación de un sistema nuevo.
- Actualización de un sistema funcionando.
- Entornos de pruebas.

3.5.1. Diseño de arquitectura

Las aplicaciones de *SIU* son sistemas web, con lo cuál la arquitectura que se propone tiene la misma naturaleza que la generada para los sistemas de aulas virtuales. En este caso, se considera desplegar todas las instancias de cada tipo de aplicación en un servidor dedicado, para que sea más simple poder conocer dónde está instalada cada aplicación. Las aplicaciones consideradas en esta instancia son las siguientes:

- *Pilaga*
- *Mapuche*
- *Guarani*
- *Diaguita*
- *Kolla*

Se puede ver en la arquitectura presentada que, al igual que lo realizado para el sistema *Moodle*, los datos se almacenan en un servidor de archivos dedicado, las bases de datos en un *cluster* de base de datos y los servidores web se instalan utilizando *Ansible*.

Debe considerarse que todas las aplicaciones anteriores están actualmente en producción, con lo cuál debe realizarse una migración de las mismas a la nueva infraestructura.

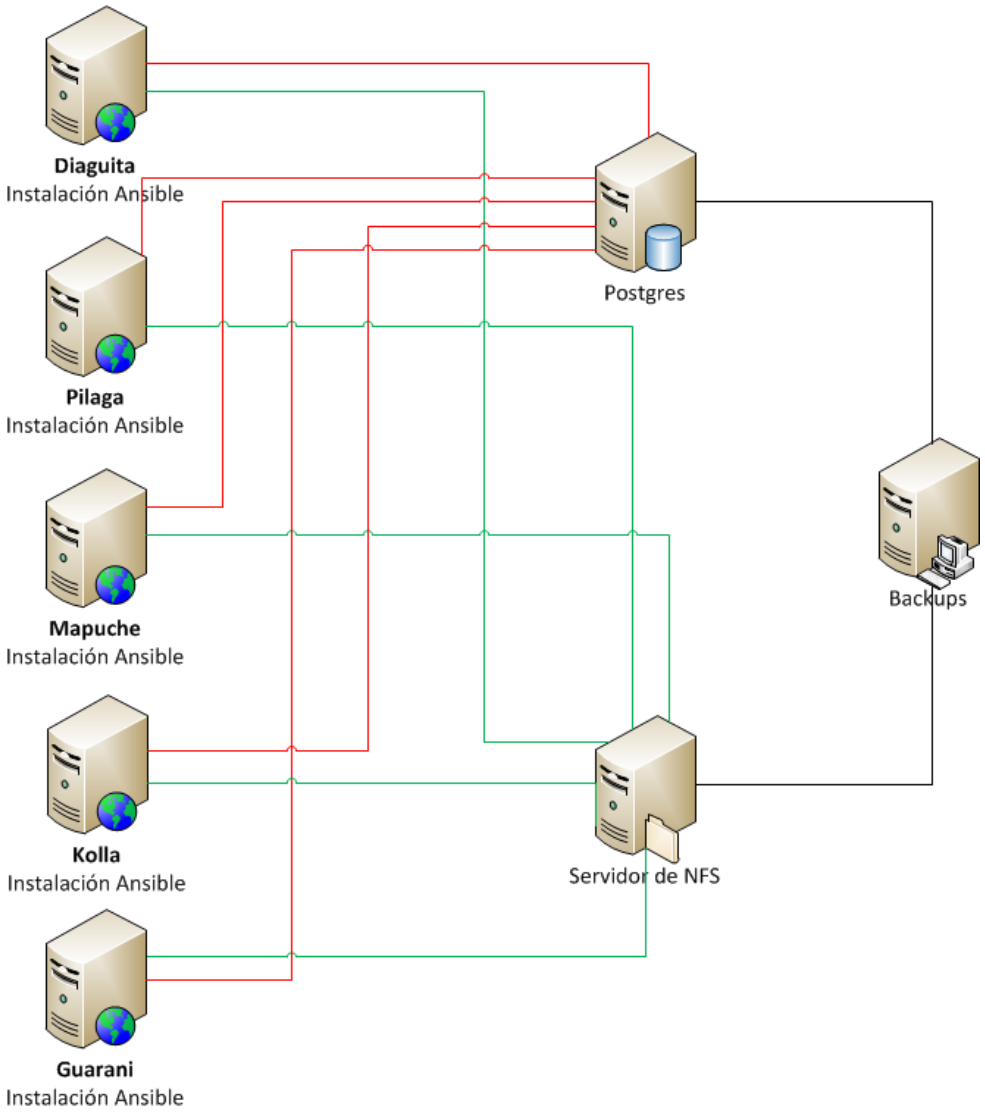

Figura 7: Arquitectura de la Infraestructura de los sistemas *SIU*

3.5.2. Implementación

En esta instancia, luego de haber implementado el *Playbook* que realiza la instalación del *Moodle*, se detecta que dicho *Playbook* puede mejorarse de manera de lograr una versión más genérica del mismo que permita ser reutilizada para la instalación de los servidores que brindan los sistemas *SIU*. En este sentido, se incorpora la utilización de *roles* que provee la herramienta *Ansible*.

A continuación se muestra cómo se adapta el *Playbook* para la instalación de *Diaguita*.

```
- hosts: all
  sudo: true
  vars_files:
    - vars/all.yml
  roles:
    - server
    - apache
    - php
    - nfs
```

Donde todas las configuraciones específicas fueron definidas por medio de variables, dicho archivo contiene:

```
---
server:
    install: '1'
    packages: [ntpdate, sudo, nfs-common, zip, locales]
    timezone: America/Argentina/Buenos_Aires
    locale: es_AR.UTF-8
    hostname: ejemplo1
    ip: 192.XX.XX.XX
    netmask: 255.255.0.0
    broadcast: 192.XX.XX.XX
    network: 192.XX.XX.XX
    gateway: 192.XX.XX.XX
    nameserver: 192.XX.XX.XX

apache:
    install: '1'
    serveradmin: ejemplo1@uner.edu.ar
    servername: ejemplo1.uner.edu.ar
    serveralias: www.ejemplo1.uner.edu.ar
    docroot: /misdatos/ejemplo1/aplicacion/www
    allowfrom: "200.XX.XX.XX/255.0.0.0"
    apptoba: ejemplo1
    pathtoba: "/misdatos/local/ejemplo1"
    tobainstancia: produccion
    pathlog: "/misdatos/log"
    instaladores: '0'
```

```
27      aliasinstaladores: "actualizar"
28      pathinstaladores: "/misdatos/instaladores"
29
30  php:
31      install: '1'
32      packages: [php5-mcrypt, php5-imagick, php5-gd, php5-pgsql]
33      install: '1'
34      max_input_vars: 15000
35      cookie_httponly: 1
36      allow_url_include: Off
37      allow_url_fopen: Off
38      memory_limit: 256M
39
40  nfs:
41      install: '1'
42      name: '/misdatos/'
43      src: '192.XX.XX.XX:/datos/siu-ejemplo1'
44      fstype: 'nfs4'
45      opts: 'proto=tcp,port=2049,_netdev'
```

El *Playbook* anterior se aplicó sobre un servidor con *Debian*, dando los resultados esperados para el sistema *Diaguita*. Luego, utilizando los mismos roles y solamente cambiando los valores de las variables, se ejecutó el mismo *Playbook* y se obtuvo la instalación de un sistema *Pilaga*, también de manera correcta.

Los cambios en las configuraciones de *PHP* se parametrizaron para evitar todo tipo de modificación manual. Así, se obtuvo un registro de toda la historia de los cambios realizados, indicando para cada cambio, quién lo realizó y en qué momento, en el servidor de control de versiones GIT.

3.5.3. Ambientes de desarrollo y testing

Debido a la necesidad de realizar pruebas con las mismas configuraciones de los sistemas en producción, se definió un procedimiento para montar un ambiente de desarrollo en base a la definición dada al inicio de este capítulo[29].

La práctica empleada antes de la creación de este ambiente de desarrollo era trabajar sobre el propio servidor de producción, copiando la carpeta de

[29]Ver la sección [3.1.1] en la página 27 para más información sobre la definición de los distintos tipos de ambientes.

instalación del sistema y creando una nueva base de datos asignada a esta instancia de réplica con una copia de los datos de producción. Si bien llevar adelante esta tarea es trivial, tiene muchas desventajas asociadas, algunas de las cuáles se listan a continuación.

- **Acceso al servidor:** un entorno de desarrollo puede requerir que un desarrollador se conecte al servidor con privilegios especiales, lo cuál no es deseable sobre un servidor productivo.
- **Seguridad:** las aplicaciones en etapa de pruebas pueden tener muchas vulnerabilidades que pueden llevar a que el resto de los sistemas en producción se vean comprometidos.
- **Integridad de los sistemas:** llevando adelante un desarrollo puede ocurrir que en el servidor se generen problemas en alguno de los demás sistemas. Por ejemplo, podría borrarse un directorio o una base de datos por error.
- **Uso de recursos:** debe garantizarse el correcto funcionamiento de los sistemas productivos. Si las aplicaciones en producción se replican en un servidor para lograr instancias de prueba, la demanda de recursos de hardware, como CPU y memoria, aumentará, pudiendo degradar la prestación de los servicios productivos.

Con la implementación presentada, usando tecnologías de automatización y aprovisionamiento y separando los datos persistentes de las configuraciones de sistema, es posible replicar los equipos en producción para crear un entorno de pruebas e incluso un ambiente de desarrollo en una estación de trabajo.

Se evaluaron varias herramientas, de las que puede encontrarse una descripción más detallada en los anexos. En un principio los ambientes locales se realizaban con *Vagrant*, pero debido a la alta demanda de recursos y al tiempo necesario para ejecutarse, se reemplazó por *Docker*[30].

Siempre en vistas de poder reutilizar el mismo *Playbook* de *Ansible* que instala y configura al servidor de producción, se busca un mecanismo que permita combinar el uso de ambas tecnologías. Así, se crea un archivo *Dockerfile* que contiene la instrucciones para crear un contenedor:

[30]Ver la sección [A.3.4] en la página 86 para más información sobre Docker.

```
FROM williamyeh/ansible:debian8-onbuild

ADD ansible /ansible
WORKDIR /ansible

# Corre Ansible para configurar la imagen Docker
RUN ansible-playbook toba-web.yml -c local

WORKDIR /data
VOLUME ["/data"]

EXPOSE 80
COPY start.sh /start.sh
CMD ["/start.sh"]
```

El archivo anterior crea una imagen de *Docker* que ejecuta el mismo *Playbook* que se utiliza para el ambiente de producción, obteniendo como resultado una configuración idéntica. No obstante, para poder reproducir el mismo entorno de producción, se debe crear otro contenedor que cumpla la tarea del servidor de base de datos. Esto se resuelve de forma muy simple utilizando *Docker*, como puede verse a continuación, dónde se crea un contenedor con *Postgres*:

```
docker run -d --name postgres-9.4 \
  -e POSTGRES_PASSWORD=testing \
  -p 5432:5432 postgres:9.4.4
```

Para finalizar con la puesta en funcionamiento del entorno de desarrollo, se debe considerar que los archivos de cada sistema se almacenan en un servidor de archivos separado. Por ello, se lleva adelante la restauración de los datos desde un backup correspondiente al servidor de producción. Esta tarea sirve, además, como prueba del correcto funcionamiento de los backups.

Finalmente, se inicia el contenedor para utilizar el ambiente de desarrollo:

```
docker run -d --link postgres-9.4:db \
  -p 8080:80 -v `pwd`/toba-data/:/data \
  toba-web
```

Aquí surge un nuevo problema que es de qué manera gestionar, de forma centralizada, los cambios que puedan tener estos contenedores y que los desarrolladores siempre tengan disponible la última versión con los cambios más

recientes. Se puede utilizar el mismo sistema de versionado GIT para guardar las recetas de construcción de los contenedores, pero la reconstrucción en forma local requiere de un conocimiento más avanzado en la utilización de la herramienta *Docker* y puede llevar un tiempo considerable, ya que se deben descargar y configurar varios paquetes de software.

Existe una manera de almacenar las imágenes ya implementadas de los contenedores y consiste en utilizar lo que se denominan *registries*, que son repositorios con las imágenes de los contenedores. El servicio que ofrece el *registry* público *Docker Hub*, tiene la ventaja de que, a partir de un *Dockerfile* que se sube a dicho repositorio, se construye automáticamente la imagen, quedando disponible para su descarga desde Internet. De esta forma, es accesible para cualquier miembro del equipo.

Es importante considerar que la imagen no debe contener ninguna información sensible y que sea lo más genérica posible, de manera que pueda ser utilizada para cualquier otro proyecto relacionado con los sistemas *SIU*.

3.6. Casos de análisis e implementación

3.6.1. Alojamiento web

Entre los servicios que provee la *DGTIC de la Universidad Nacional de Entre Ríos* se puede mencionar el de alojamiento de sitios web, muchas veces desarrollados con plataformas de código abierto muy populares como es el caso de *Wordpress* y de *Drupal*. Estas plataformas, escritas en *PHP*, requieren un servidor web, el intérprete de código *PHP* y un motor de bases de datos como es el caso de *MySQL*. Debido a que los requerimientos son similares y, para lograr un mejor aprovechamiento del hardware del que se dispone, varios de los sitios se instalan en un mismo servidor utilizando diferentes *hosts virtuales*.

Estos sistemas, que son de acceso público, suelen tener muchas vulnerabilidades[31] que llevan a que la seguridad de alguno de ellos pueda verse comprometida, pudiendo extenderse el impacto de esta situación al resto de los sitios web alojados en el mismo servidor. Adicionalmente, si bien en la generalidad de los casos los requerimientos son similares, se da la situación

[31]Drupal tuvo 290 vulnerabilidades en 13 años, como puede verse en http://www.cvedetails.com/vendor/1367/Drupal.html. Por su parte, Wordpress suma 245 vulnerabilidades en 11 años http://www.cvedetails.com/vendor/2337/Wordpress.html.

en la cuál los sistemas necesitan diferentes versiones del intérprete de código, *PHP* en este caso. Esto genera un inconveniente para el administrador, debido a que sistemas operativos como *Debian* no permiten instalar dos versiones diferentes de *PHP* haciendo uso de los paquetes que provee el propio sistema operativo. La forma de poder ejecutar diferentes versiones de *PHP* en el mismo servidor es haciendo una instalación manual de cada versión; esto implica que el administrador compile cada versión de *PHP* que los sistemas necesiten. Si bien entonces es posible lograrlo, la dificultad surge al momento de administrar estas instalaciones, sobre todo en lo relativo a la aplicación de parches de seguridad o a la incorporación de nuevos módulos o funcionalidades.

Teniendo en cuenta los problemas planteados en el párrafo anterior es que se buscó un mecanismo para solucionarlos y así fue como se tomó la decisión de utilizar *Docker*[5][4]. La propuesta para solucionar los problemas planteados consiste en que cada aplicación se ejecute en su propio contenedor de *Docker*. De esta manera, se obtienen las siguientes ventajas:

- Una aplicación se ejecuta en un entorno en el que tiene sólo el software que necesita, sin tener en el sistema librerías o servicios superfluos que pudieran exponer vulnerabilidades.
- Se logra aislar todas las aplicaciones entre sí, lo que limita el impacto de una vulnerabilidad de seguridad en una aplicación al contenedor en que se ejecuta, evitando el impacto en el resto del sistema.
- Como el entorno es propio y dedicado a la aplicación, se podría disponer de distintos contenedores cada uno con un sistema operativo diferente, lo que permite que en un mismo host haya aplicaciones ejecutándose sobre, por ejemplo, *Ubuntu* 12.04 y otras haciéndolo sobre *Debian* 7. Esto soluciona el problema de contar con distintas versiones de *PHP* en el mismo servidor.
- Un desarrollador puede programar su aplicación en un contenedor que sea probado hasta asegurarse que funciona como se desea y es una copia idéntica de ese contenedor el que luego se ejecutará en producción, garantizando así que no existirá ningún problema en el pasaje a producción de una aplicación. Incluso, en caso de necesitar modificar una aplicación, el desarrollador puede tomar la imagen del contenedor que está en producción, trabajar sobre la misma, guardar los cambios y poner esa imagen en producción luego de que todo funcione como se espera.

En base a lo analizado hasta este punto, se diseñó la siguiente arquitectura:

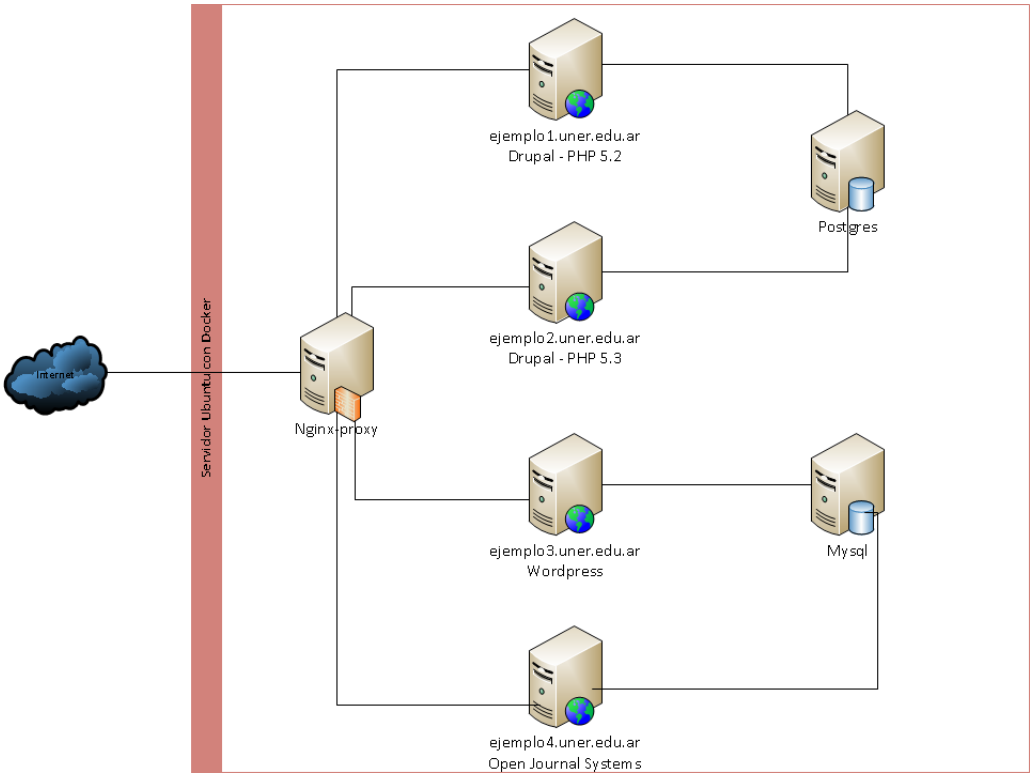

Figura 8: Arquitectura *Docker* de Web Hosting

Todos los contenedores que se crearán, algunos de los cuáles se muestran a continuación, serán ejecutados sobre un servidor con *Ubuntu*.

Para crear el contenedor con un motor de bases de datos *mysql*:

```
$ docker run --restart=always --name mysql-5.5 \
  -v `pwd`/datadir:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=XXXX -d mysql:5.5
```

Para crear el contenedor con un motor de base de datos *postgres*:

```
$ docker run --restart=always --name postgres-9.2 \
  -v `pwd`/datadir:/var/lib/postgresql/data \
  -e POSTGRES_PASSWORD=XXXX -d postgres:9.2
```

El contenedor que maneja los *hosts virtuales* utiliza *Nginx*[11]. Para ello se hace uso de una imagen de *Docker Nginx* ya existente[6] que, por medio de una variable, se comunica con los demás contenedores a los que debe hacer la redirección del sitio:

```
$ docker run --restart=always --name nginxproxy -d \
  -p 80:80 -v /var/run/docker.sock:/tmp/docker.sock:ro \
   jwilder/nginx-proxy
```

Luego, para poner operativo un sitio con *Drupal*:

```
$ SITIO=ejemplo1.uner.edu.ar
$ docker run --restart=always -d --link postgres-9.2:postgres \
  -e VIRTUAL_HOST=$SITIO,www.$SITIO --name $SITIO \
  -v "$PWD"/data:/var/www/html drupal-postgres
```

Si lo que se necesita es un sitio que ejecute *Wordpress*:

```
$ SITIO=ejemplo2.uner.edu.ar
$ docker run --restart=always -d --link mysql-5.5:mysql \
  -e VIRTUAL_HOST=$SITIO,www.$SITIO --name $SITIO \
  -v "$PWD"/data:/var/www/html wordpress-uner
```

De forma similar, el mismo servidor de contenedores tiene otros quince sistemas diferentes, cada uno ejecutando en su propio contenedor, lo que evidencia una gran consolidación de servicios y de ambientes heterogéneos conviviendo en perfecta armonía en un mismo servidor.

3.6.2. Servicio de DNS

El servicio de DNS, si bien es uno de los más simple de administrar, es también uno de los más críticos, debido a que un fallo en este servicio provoca normalmente que dejen de funcionar todos los demás. Es, a su vez, un servicio dinámico, debido a que la información de su base de datos se modifica en cada oportunidad en la que se instancian nuevas aplicaciones, se despliegan nuevos servicios o incluso se modifican o dan de baja sistemas y/o servicios existentes.

Debido a que las entradas del DNS se configuran por medio de archivos de texto plano[32], que suelen modificarse directamente en el servidor, resulta difícil poder mantener un registro de los cambios efectuados. Asociado a esto, si por error se elimina y/o modifica algún registro, es difícil obtener la información que existía anteriormente en los archivos de configuración. En esta situación puede evidenciarse uno de los casos típicos de la gestión y administración de los servidores a los que se hace referencia en la introducción de este trabajo[33].

Por lo explicado en los párrafos anteriores y con el objetivo de poder mostrar la potencialidad de las herramientas de automatización, combinadas con el versionado de la infraestructura, se tomó el sencillo caso del DNS, se diseñó un proceso de gestión del servicio y se escribió un *Playbook* para administrarlo.

3.6.2.1 Proceso de gestión del DNS

La gestión tradicional del DNS, como se explicó antes, implica editar directamente en producción los archivos de configuración y reiniciar o recargar el servicio. Al ser una tarea manual, es común omitir la verificación del archivo editado. En caso de que el mismo contenga errores[34], el resultado será que el servicio deje de funcionar correctamente, situación que es posible que no sea advertida por el administrador hasta que surja algún inconveniente o como consecuencia del reclamo de algún usuario[35].

Se propone entonces un proceso de gestión del DNS utilizando herramientas de automatización y versionado, en el primer caso para evitar los errores humanos que puedan surgir por omisión y/o distracción, en el segundo caso, para mantener registro de todos los cambios introducidos en la infraestruc-

[32]En el caso de la *Universidad Nacional de Entre Ríos* se utiliza el software *BIND* que tiene esta metodología de configuración.

[33]Ver la introducción en la página 6.

[34]En todas las implementaciones de *BIND* testeadas sobre Ubuntu el servicio no realiza ningún chequeo de los archivos de configuración de cada zona. Si algún archivo de una zona contiene un error, el servicio no carga la definición de la zona, lo que implica que no resuelve ningún nombre asociado a dicha zona y esto lo hace sin devolver ningún error en la salida estándar, sólo puede verse en los logs del servicio.

[35]Cabe destacar que el administrador tiene posibilidades de verificar el funcionamiento del servicio. No obstante, la posibilidad de que omita dicha verificación existe y por ese motivo se la considera como una posibilidad.

tura.

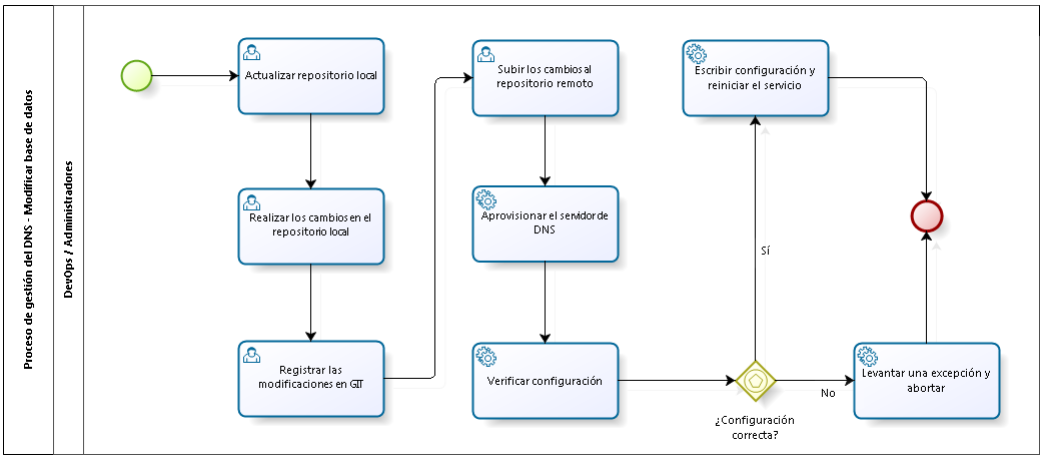

Figura 9: Proceso de gestión del DNS

En el proceso descripto en la figura anterior pueden verse una serie de tareas que se detallan debajo.

- **Actualizar repositorio local:** en el contexto del proceso de gestión del DNS, el administrador de sistemas tiene en su estación de trabajo una copia del repositorio versionado con GIT con las configuraciones del servidor de DNS. En esta tarea, lo que debe hacer dicho actor es actualizar su propio repositorio local con la última información existente en el repositorio remoto.
- **Realizar los cambios en el repositorio local:** en esta tarea se realizan las modificaciones a los archivos de zona del DNS. En otras palabras, es en este punto donde se agregan, modifican y/o eliminan registros del dominio implicado.
- **Registrar las modificaciones realizadas en GIT:** una vez que se realizaron las modificaciones pertinentes, las mismas deben ser registradas. Esto se hace efectuando un *commit* en el repositorio local de GIT.
- **Subir los cambios al repositorio remoto:** las modificaciones ya registradas en el repositorio local se suben al repositorio remoto de GIT, desde donde serán visibles para el resto del equipo autorizado.

- **Aprovisionar el servidor de DNS:** hasta este punto las tareas fueron realizadas manualmente por el administrador. En esta instancia en particular, el mencionado actor lanza la tarea de aprovisionamiento del servidor de DNS que dispara un proceso que copia los archivos de zonas al servidor de DNS remoto, pudiendo dejar los archivos por ejemplo en un directorio temporal.

- **Verificar configuración:** el proceso automatizado de aprovisionamiento y configuración toma los archivos de configuración desde el directorio donde fueron cargados y verifica que la sintaxis de los mismos sea correcta. Este punto es de gran importancia, debido a que asegura que en caso de existir un error en la configuración provista por el administrador los cambios no serán aplicados, garantizando que no se interrumpirá el servicio. Notar cómo, al reemplazar dicha verificación, que era manual en el proceso tradicional, por una automatizada en el proceso propuesto, se logra eliminar el riesgo asociado al error humano mencionado anteriormente como de omisión y/o distracción.

- **Levantar una excepción y abortar:** esta tarea depende del resultado de la verificación anterior. Si la misma falló, entonces el proceso termina levantando una excepción que permitirá al administrador saber que existió un error, pudiendo entonces volver a ejecutar el proceso corrigiendo el error introducido.

- **Escribir configuración y reiniciar el servicio:** en el caso que la verificación haya sido exitosa, se reemplaza finalmente la configuración del DNS por la nueva provista y se reinicia o recarga el servicio para que los cambios se apliquen.

Hasta aquí se vio el proceso de gestión del DNS propuesto. A continuación se abordará la implementación de este proceso con una solución concreta utilizando *Ansible*.

3.6.2.2 Gestionando el DNS con *Ansible*

La receta que se escribió es muy simple, se basa en la configuración preexistente del servicio:

```
1 - hosts: all
2   tasks:
3    - name: Actualizando los repositorios
```

```
4       shell: apt-get update
5
6     - name: Instalando utilidades como rsync, bind9
7       action: apt pkg={{item}} state=present
8       with_items:
9        - rsync
10       - bind9
11
12    - name: Sincronizando los cambios
13      synchronize: src=files/bind/ dest=/etc/bind/
14
15    - name: Seteando permisos
16      file: path=/etc/bind/ owner=root group=bind recurse=yes
17
18    - name: Seteando permisos
19      file: path=/etc/bind/secundario/ owner=bind recurse=yes
20
21    - name: Seteando permisos
22      file: path=/etc/bind/rndc.key owner=bind mode=640
23
24    - name: Chequeando que los cambios esten OK
25      shell: /usr/sbin/named-checkzone uner.edu.ar \
26             /etc/bind/maestro/uner.edu.ar
27
28    - name: Reiniciando el servicio
29      shell: /etc/init.d/bind9 restart
```

La nueva forma de manejar el DNS consiste en agregar una entrada en el archivo correspondiente en la máquina local y luego ejecutar:

```
$ ansible-playbook -u root dns.yml -i inventories/
```

Una vez efectuados los cambios, se deben registrar en el sistema de control de versiones como se muestra a continuación.

```
$ git add files/bind/maestro/uner.edu.ar

$ git commit -m "Nueva entrada ejemplo1.uner.edu.ar"

$ git push
```

Con las acciones anteriores pueden verse claramente luego los cambios a través de la interfaz web del repositorio de GIT.

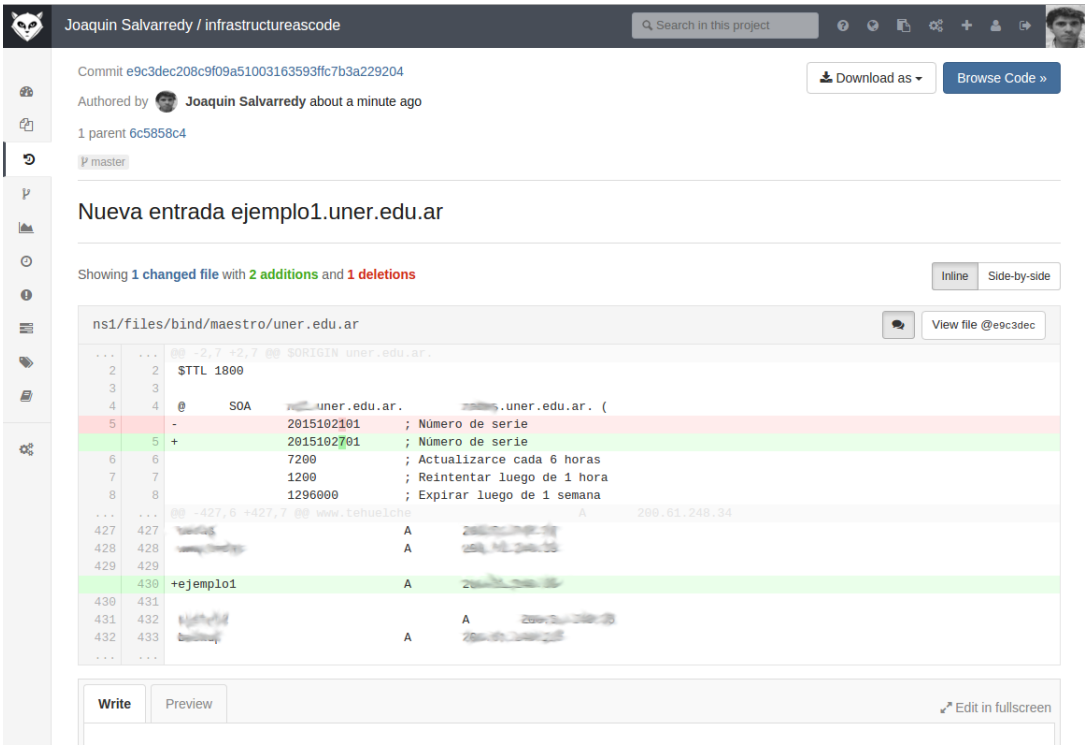

Figura 10: Entrada de DNS en *Gitlab*

Como se puede observar en la imagen, se visualiza perfectamente quién realizó las modificaciones y la entrada que se agregó. Esta manera de gestionar el DNS permite tener la seguridad de que los cambios se realizan de manera correcta, pudiendo siempre volver a una versión anterior de la configuración de la zona del DNS.

En lo que respecta a las copias de seguridad, lo crítico en este punto es el repositorio del sistema de control de versiones, que es el que debe resguardarse, debido a que la instalación y la configuración del servidor de DNS se realiza haciendo uso del *Playbook* correspondiente. Incluso así, por la naturaleza de GIT, es probable que en todo momento existan copias del repositorio en las máquinas de los administradores.

Como reflexión final sobre la implementación de este servicio, vale la pena comentar que el servidor de DNS se encontraba en una máquina virtual dedicada y, por la criticidad del servicio, se realizaban *snapshots* de forma periódica, ocupando cada uno de esos *snapshots* alrededor de 750 Megabytes de espacio en disco. Con la redefinición del servicio se redujo dicho espacio a unos 750 Kilobytes, que es lo que ocupan los archivos de configuración junto con el *Playbook* que instala el servidor. Así, no sólo se redujo al uno por mil

el espacio ocupado, sino que además se obtuvo una historia completa de los cambios de la que antes no se disponía.

4. Capítulo IV

4.1. Análisis

En esta sección se presentan algunos comentarios respecto de los temas tratados a lo largo del trabajo, con conclusiones respecto de varios de ellos luego de haberlos estudiado y con las experiencias derivadas de las implementaciones realizadas, acompañadas del uso de algunas de las herramientas mencionadas.

4.1.1. El problema de las metodologías ágiles

En comparación con las metodologías tradicionales, las metodologías ágiles aportaron a los equipos de desarrollo de software una dinámica de trabajo vertiginosa, lo que posibilitó la obtención de productos entregables en plazos menores de tiempo. De la misma manera, la liberación de nuevas versiones de software pasó de grandes lanzamientos cada varios meses o incluso años a la introducción de pequeños cambios cada pocas semanas. Esta forma de trabajo fue adoptada prácticamente por todas las empresas que producen software.

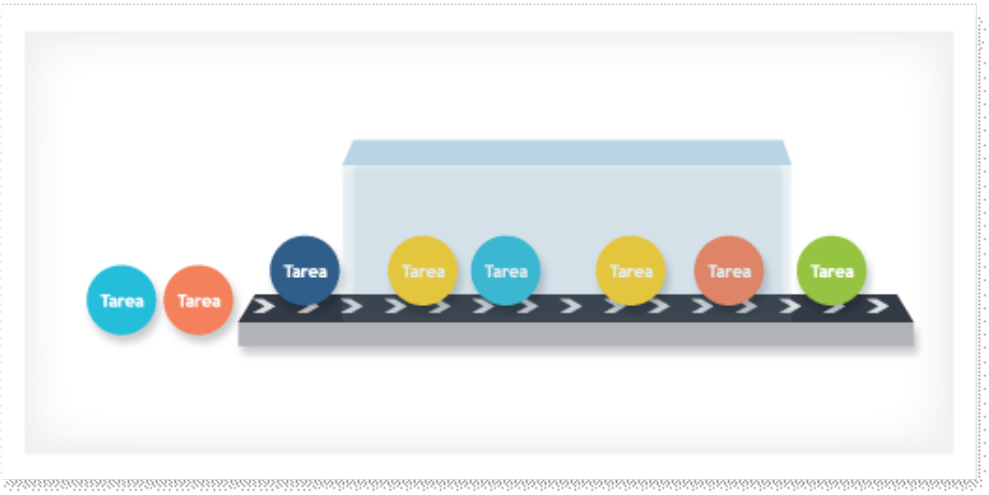

Figura 11: Flujo de trabajo tradicional entre desarrollo y operaciones

Este cambio en la manera de gestionar los proyectos de software y los

grupos de trabajo, que tan buenos resultados dio desde el punto de vista del desarrollo, se convirtió en un problema de relevancia para el área de operaciones y de administración de sistemas, que no estaba preparada para absorber la demanda constante de cambios que comenzó a recibir. Así fue como el área de operaciones se transformó rápidamente en un factor limitante para el despliegue de nuevas funcionalidades en las aplicaciones.

El problema en este punto es que las metodologías ágiles enfocaron su esfuerzo en optimizar y agilizar los procesos de desarrollo de software y las organizaciones que las adoptaron perdieron de vista el proceso completo que incluye al despliegue de las aplicaciones en producción. Optimizar un proceso implica trabajar sobre el proceso en conjunto[36] para que no se generen cuellos de botella en algún punto que, no sólo no permita que la optimización realizada mejore el proceso sino que además termine saturando algún área, lo que deriva en que el trabajo acumulado en esa área crecerá indefinidamente hasta el punto en que no sea posible resolver el trabajo pendiente.

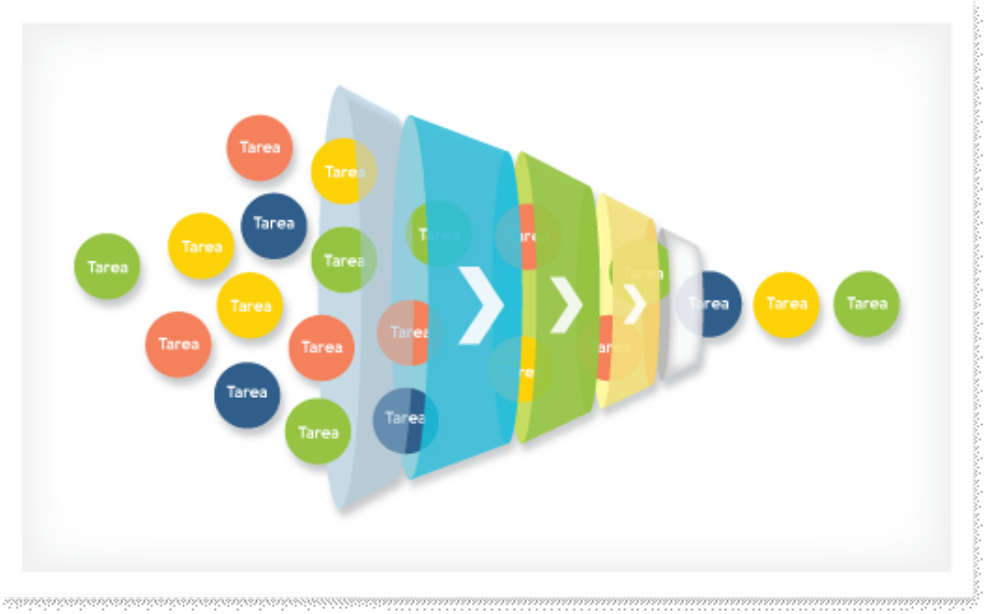

Figura 12: Metodologías ágiles en el desarrollo, antes de DevOps

Esta situación comenzó a ser evidente en muchas organizaciones que entendieron que era necesario optimizar el área de operaciones y así fue como comenzaron a incorporarse herramientas que las metodologías ágiles habían adoptado para el desarrollo de software en las mencionadas áreas de admi-

nistración de sistemas, hasta alcanzar como punto final la implementación de las metodologías ágiles pero orientadas específicamente a la administración de sistemas a través del movimiento de *DevOps*.

Figura 13: Flujo de trabajo con DevOps

En este trabajo, se inició el camino incorporando al área de operaciones de la *DGTIC de la Universidad Nacional de Entre Ríos* la metodología *Kanban* para luego implementar una forma de trabajo basada en *DevOps*. A continuación se harán algunos comentarios finales sobre estos dos aspectos mencionados.

4.1.2. Kanban en el área de operaciones

Como consecuencia de la investigación realizada en este trabajo y la observación y análisis de las actividades realizadas en la *DGTIC de la Universidad Nacional de Entre Ríos*, se decidió adoptar la metodología *Kanban*. Con esta decisión, se buscó lograr una mejora en el flujo de trabajo del área de administración de sistemas y optimizar la organización de sus tareas.

Esta metodología, que se aplicó principalmente para la gestión de proyectos, permitió mejorar la productividad al hacer visible en un tablero el

trabajo en progreso junto con las tareas pendientes de realización y las completadas. Desde dicho tablero, se pueden gestionar las tareas de forma simple e intuitiva. Otra característica clave en este sentido es poder expresar la dependencia entre tareas, lo que permite fácilmente saber, de las tareas pendientes, cuáles pueden ejecutarse de manera concurrente y cuáles deben aguardar la finalización de otra para poder iniciarse.

Finalmente, la flexibilidad que brinda *Kanban* al poder incorporar nuevo trabajo al ya planificado es de especial importancia en un área de operaciones, donde por la naturaleza del trabajo no es posible siempre prever todas las tareas que será necesario realizar en un determinado período de tiempo. En este sentido, otras metodologías más rígidas no se adaptan de forma correcta a las necesidades del área.

4.1.3. DevOps y la infraestructura como código

Hasta aquí, se plantearon una serie de cuestiones en el área de operaciones que se repasan a continuación.

- Problemas asociados a la implementación de metodologías ágiles en las áreas de desarrollo.
- Necesidad de poder replicar ambientes en pocos minutos.
- Minimización del impacto entre el desarrollo y la puesta en producción.
- Plan de contingencia que permita salir adelante en caso de enfrentarse a situaciones críticas.

Ante la perspectiva planteada, se decidió optar por la implementación de *DevOps* y la utilización de infraestructura como código. Si bien las ventajas de la adopción de *DevOps* resultan atractivas para un administrador de sistemas, el trabajo de llevar la teoría a la práctica no es trivial y demanda un esfuerzo de parte de los administradores de sistemas, junto con un acompañamiento que resulta necesario y prácticamente excluyente por parte de los directivos. Las problemáticas son muy variadas y algunas de ellas se resumen debajo.

- **Tiempo necesario de implementación:** implementar la gestión de la infraestructura con un enfoque de *DevOps* implica un trabajo de desarrollo de software y, como tal, exige planificación, diseño, desarrollo y pruebas. Poder implementar toda la infraestructura con esta

metodología requiere tiempo y el equipo de TI debe contar con el apoyo de las autoridades en este sentido.

- **Curva de aprendizaje:** es quizá uno de los factores más restrictivos a la hora de enfrentar un proyecto de *DevOps*. Dependiendo de la solución que se busque y la herramienta que se elija puede ser más o menos pronunciada pero, en cualquier caso, existe una dificultad inherente, particularmente asociada al cambio radical en la forma de trabajar y a la adopción de múltiples herramientas nuevas.

- **Resistencia al desarrollo:** lo natural es que en cualquier organización el enfoque de *DevOps* sea llevado adelante principalmente por administradores de sistemas que son los responsables de los servidores y de la prestación de los servicios. Ocurre que, en general, los administradores de sistemas no están habituados al trabajo de desarrollo de software, menos aún para instalar sus propios servidores y esto puede generar cierta resistencia.

- **Tentación de trabajar directamente sobre el servidor:** la introducción de cambios en la infraestructura puede requerir modificar el código con el que se instaló determinado servicio y/o servidor, incluso cuando sólo se necesite modificar una línea. Esto implica un trabajo adicional y puede llevar al administrador a verse tentado a aplicar los cambios directamente sobre el servidor. Dicha situación supone un problema significativo cuando se utiliza una metodología de *DevOps*, debido a que esos cambios muy probablemente serán sobreescritos más tarde por lo que indique el código.

- **Superar la frustración:** *DevOps* cambia radicalmente la forma de trabajar en las áreas de operaciones. Como ya se mencionó, requiere un aprendizaje y tiene una real dificultad. Esto hará inevitable que las cosas no siempre salgan bien o que el esfuerzo pueda parecer en vano. Incluso un administrador experimentado se enfrentará constantemente a la realidad de sentir que ya no es capaz de cumplir con su trabajo. Esto puede llevar a la frustración de dicha persona y, eventualmente, a abandonar la implementación de la metodología. En esta instancia, es importante la constancia y el apoyo de los compañeros y las autoridades.

4.1.4. Información y persistencia

Cuando se trabaja con la concepción de tratar a la infraestructura como código, algo que debe analizarse particularmente es qué es información y, más específicamente incluso, qué de todo lo categorizado como información es información que debe persistirse[36]. En este sentido se debe considerar una serie de aspectos en relación a los servidores y a las aplicaciones que se brindan. En ese orden se detallan a continuación.

En referencia a los servidores y servicios, deben considerarse, a grandes rasgos:

- Paquetes de software y librerías instaladas.
- Configuraciones generales y específicas del servidor.
- Usuarios y permisos.
- Configuración de cada servicio prestado.

Las aplicaciones, por su parte, tienen en líneas generales:

- Dependencias de servicios y librerías.
- Código de la aplicación.
- Configuraciones específicas.
- Datos almacenados en bases de datos.
- Archivos y directorios incorporados, como pueden ser los archivos que se suben a una página web.

Al gestionar la infraestructura como código, ya sea haciendo uso de alguna herramienta de aprovisionamiento o bien utilizando sistemas inmutables (o mejor aún, con la combinación de ambas cosas), la gran mayoría de los ítems mencionados se pueden considerar persistentes sin realizar ningún trabajo adicional. Repasando, cabe recordar que los sistemas de aprovisionamiento permiten:

[36]En este apartado, se entiende por persistencia a la condición que hace que la información sea durable en el tiempo, considerando incluso para ello las copias de respaldo necesarias.

- Instalar software en un servidor.
- Configurar todos los aspectos necesarios, tanto del servidor como del software y las aplicaciones instaladas.
- Definir usuarios, crear estructura de directorios, asociar permisos y niveles de autorización.
- Especificar una serie de pasos a realizar para, por ejemplo, instalar una aplicación. Esto hace posible explicitar todas las dependencias que una aplicación tenga y asegurarse que las mismas estén disponibles antes de instalar la aplicación.

Con lo anterior, puede notarse que se estarían cubriendo prácticamente todos los puntos planteados. Sin embargo, quedan aún analizar la persistencia de tres elementos muy importantes de las aplicaciones que son:

1. Código de la aplicación.
2. Información almacenada en las bases de datos.
3. Archivos y directorios dinámicos de la aplicación.

Existe una diferenciación muy evidente entre el punto 1 y los puntos 2 y 3 de la lista anterior y es que el 1 es generalmente estático y sus cambios son claramente identificables y gestionados por el personal de la *DGTIC de la Universidad Nacional de Entre Ríos*. En cambio, los puntos 2 y 3 son dinámicos y el cambio suele quedar en manos de terceros y se producen sin conocimiento de la *DGTIC de la Universidad Nacional de Entre Ríos*, como puede ser el caso de un comunicador social cargando información en un sitio web.

Solucionar la persistencia del código de la aplicación es simple y, trabajando con un enfoque de *DevOps*, suele estar ya considerada por la misma lógica de la instalación de los sistemas, que normalmente consiste en obtener el código de un repositorio donde el mismo está alojado, como podría ser un repositorio *GIT*.

En cambio, no es posible abarcar con las herramientas de aprovisionamiento la información almacenada en la base de datos y los archivos dinámicos de la aplicación, por tratarse precisamente de datos dinámicos. Por tal motivo, estos dos puntos deben considerar un esquema tradicional de copias de respaldo.

Vale mencionar también que hasta aquí siempre se dio por sentado que toda la información que define la infraestructura necesaria, entiéndase aquella información de la que se valen las herramientas de aprovisionamiento para ejecutar su trabajo, existe en algún sitio. Si bien eso es cierto, es fundamental tener en cuenta que esa fuente de información **debe ser resguardada** para asegurar la persistencia de aquellos puntos que se asumieron garantizados por las herramientas de aprovisionamiento.

4.1.5. Estandarización y gestión de cambios

Una tarea compleja de llevar adelante es la de estandarizar la instalación de servidores, servicios, aplicaciones, entre otras. Incluso lograda una estandarización, es difícil también garantizar que la misma se sostenga en el tiempo. Así por ejemplo, pueden encontrarse situaciones donde, dependiendo de quién haya instalado una aplicación, la misma se ubique en los directorios `/var/www`, `/usr/local`, `/opt` o alguna otra ubicación que quién haya instalado la aplicación haya considerado apropiada en ese momento. Estas prácticas atentan contra la predictibilidad de la infraestructura y dificultan su administración.

Al contar con esquemas de versionado de configuraciones, es posible establecer la trazabilidad de los cambios con su correspondiente documentación y disponer de la capacidad de volver atrás modificaciones que pudieran haber introducido problemas. Por otro lado, las herramientas de automatización y aprovisionamiento de la infraestructura, brindan la posibilidad de obtener sistemas homogéneos, sin depender de quién haya instalado cada servidor.

4.2. Limitaciones

En base a lo analizado hasta aquí, se encuentran una serie de limitaciones que no se cubren en este trabajo, pero que deben considerarse como parte de un proceso de gestión integral.

4.2.1. Restauración masiva de la infraestructura

En este trabajo se analizó la forma de instalar, gestionar, mantener y restaurar servidores completos, considerando incluso las aplicaciones de estos. No obstante, ante una falla masiva de la infraestructura, por la necesidad de replicarla o por cualquier otra razón que involucre restaurar la infraestruc-

tura completa, existen algunos problemas que deben tenerse en cuenta para diseñar y planificar la manera de resolverlos.

- Cómo crear cada nuevo servidor.
- Orden de creación de los servidores.
- Restauración de los datos de los servidores y aplicaciones.

La manera de enfrentar esta situación es a través de documentación. Así, puede utilizarse una *wiki* donde se detalle cada uno de los comandos necesarios para restaurar cada servidor e incluso especificando el orden en que debe hacerse. No obstante, cuando se comienza a gestionar la infraestructura con herramientas de automatización y aprovisionamiento, el crecimiento del número de equipos gestionados suele incrementarse con facilidad debido a que gestionar una infraestructura de gran tamaño se simplifica mucho. Esto implica gran cantidad de equipos que levantar y este aspecto solo ya transforman la tarea en lenta y compleja[37]. Además, se vuelve a enfrentar un problema ya conocido que es la imposibilidad de garantizar que la documentación refleje el estado real de la infraestructura.

En la sección de trabajos futuros se mencionan algunas soluciones a esta problemática.

4.3. Conclusiones

La motivación para llevar adelante este trabajo surge como consecuencia de una observación realizada sobre los sectores que proveen servicios de infraestructura tecnológica en diferentes organismos públicos, especialmente aquellos que tienen una interacción cotidiana con sectores de desarrollo de software, los cuáles se enfrentan a la problemática de gestionar entornos complejos y dinámicos, en la mayoría de los casos con reducidos planteles técnicos y profesionales. En este contexto, se escogió como caso particular de estudio a la *DGTIC de la Universidad Nacional de Entre Ríos*, haciendo un análisis de sus requerimientos con el objetivo de generar propuestas

[37]Existen muchos factores más a considerar como puede ser la existencia de determinados datos residuales en los repositorios centrales. Tal es el caso al utilizar Chef por ejemplo. No obstante, el objetivo en esta instancia es plantear el escenario sin entrar en detalles.

e implementaciones que pudieran colaborar para simplificar la gestión de su infraestructura.

Entendiendo a la problemática como una situación generalizada, fue que se realizó una búsqueda de posibles soluciones o formas de optimizar la administración de los servicios y se plantearon entonces algunos objetivos que guiaron el desarrollo de este trabajo. En relación a dichos objetivos es que se revisan en la siguiente sección los aportes que ha dado como resultado este trabajo.

4.3.1. Aportes realizados

Esta tesina realizó aportes en diferentes sentidos, que van desde el punto de vista de la investigación y la teoría, incluyendo aspectos prácticos que llevaron a la implementación de varias de las soluciones propuestas y planteando múltiples trabajos a futuro que pueden resultar de gran interés y utilidad, no sólo para la *DGTIC de la Universidad Nacional de Entre Ríos* sino también para otros organismos públicos y privados que puedan enfrentarse a las mismas o similares problemáticas. Dichos aportes se presentan a continuación.

- Mejora de la organización en la gestión de proyectos haciendo uso de metodologías ágiles como *Kanban* y herramientas como el caso de Kanboard para favorecer la colaboración, la comunicación y el cumplimiento de las tareas en el grupo de trabajo.
- Adopción de una metodología basada en *DevOps*, con la que se obtienen las ventajas de:
  - Versionado de la infraestructura, contando con la historia completa de los cambios en la configuración de los servidores y los servicios.
  - Configuraciones de servicios que han sido probadas y pueden replicarse cuántas veces sea necesario sin trabajo adicional y minimizando la posibilidad de errores.
  - Posibilidad de lograr una escalabilidad horizontal de las aplicaciones de forma simple y rápida, algo muy atractivo según las arquitecturas de aplicaciones que existen actualmente.
  - Sistemas homogéneos y predecibles, lo que facilita su administración.

- Mejora en la comunicación de los grupos de desarrollo y operaciones, favoreciendo la interoperabilidad de las áreas.
- La creación de recetas que instalen la infraestructura y las aplicaciones generó la necesidad de estudiar y entender mejor cómo funciona cada pieza en una arquitectura de servicios y aprender cómo están desarrolladas las aplicaciones, como por ejemplo las que utilizan el *framework SIU-Toba*. Esto no sólo favorece el desarrollo profesional y técnico del personal involucrado sino que también mejora la comprensión de los sistemas, lo que es de gran ayuda al momento de tener que solucionar problemas.
- Permitió ver cómo otros profesionales realizan la instalación de su propia infraestructura, lo que constituye un material de aprendizaje y superación constante nunca antes experimentado en lo que hace a las áreas de administración de sistemas.
- Posibilidad de replicar ambientes, servidores y servicios idénticos en pocos minutos, lo que cumple el doble objetivo de poder realizar pruebas y de poder actuar de forma más rápida, segura y predecible ante contingencias.

- Estudio del estado de arte respecto de herramientas de automatización y aprovisionamiento de la infraestructura, analizando características de cada una, ventajas y desventajas y descubriendo el potencial que puede alcanzarse con este tipo de herramientas, lo que lleva a tomar una dimensión diferente de la problemática de TI.

- Propuestas de diferentes procesos para gestionar algunas cuestiones críticas y/o problemáticas como son:

  - Solicitud de nuevas aplicaciones: se detectó que no se cuenta con información concreta y clara sobre las aplicaciones existentes en la infraestructura de la *DGTIC de la Universidad Nacional de Entre Ríos* que permita identificar y contactar al responsable de cada aplicación. Esto supone un problema al momento de tener que verificar la vigencia de una aplicación, en caso de tener que realizar un mantenimiento a la misma o cuando se detecta algún problema de seguridad o cualquier otra tarea que demande tomar decisiones respecto de una aplicación. Con este proceso, se logró plantear un esquema que garantice que se tenga de toda aplicación instalada en la infraestructura de la *DGTIC de la Universidad Nacional de Entre Ríos* la información mínima del responsable, avalada por una autoridad y con el conocimiento explícito del solicitante de

la aplicación de cuáles son las políticas con las que se presta el servicio.

- Gestión de aplicaciones en testing: los ambientes de pruebas suelen tener aplicaciones que son útiles por un período de tiempo y luego dejan de ser necesarias. El problema en este punto es que no se cuenta con información que permita para conocer en todo momento qué aplicaciones están en uso y cuáles obsoletas y esto lleva a que, con el tiempo, se tengan muchas aplicaciones en entornos de testing ocupando recursos innecesariamente. Con este proceso se buscó dar solución a este problema, con el valor agregado de que la propuesta plantea una gestión totalmente automatizada.
- Gestión de cambios en producción: otro punto crítico son los cambios que se realizan directamente en producción, llevándose a cabo con usuarios privilegiados, sin una prueba previa en un ambiente de pre-producción, e incluso siendo esos cambios ejecutados por desarrolladores. Este proceso plantea un flujo de trabajo que considera pruebas en pre-producción, elimina la necesidad de que los desarrolladores tengan acceso a los servidores e incluso propone la automatización de la mayoría de las tareas.

- Implementación de servicios y despliegue de aplicaciones utilizando tecnologías como *Ansible* y *Docker* para favorecer la gestión de la infraestructura, llevando a la práctica varios de los puntos analizados en este documento. Esto permitió a su vez, estandarizar la instalación de los servidores web y la configuración de las aplicaciones involucradas en las implementaciones llevadas adelante.

- Definición de diferentes ambientes para desarrollo, testing, pre-producción y producción para minimizar los errores en el despliegue de aplicaciones a producción y fomentar las buenas prácticas en un entorno combinado de desarrollo y producción.

- Independencia del administrador que, si bien en una metodología de *DevOps* debe incorporar nuevas habilidades, el resultado luego de aprovisionar la infraestructura no depende de quién lleve adelante dicho aprovisionamiento.

- Se perdió el "miedo" a reemplazar determinados sistemas y/o servicios, debido a la posibilidad de probar una instalación hasta poder estar totalmente seguros del funcionamiento y tener la posibilidad de replicarla luego para hacer el reemplazo definitivo.

- Planteados varios trabajos a futuro que permitirían, no sólo complementar lo realizado en el transcurso de esta tesina, sino además incrementar sustancialmente la eficiencia, la flexibilidad en la administración de la infraestructura, la capacidad de respuesta ante fallos y la gestión de aplicaciones, al tiempo que, por medio del monitoreo y las estadísticas, se podría tener una visión más global del estado de los servicios y realizar un análisis y un dimensionamiento de la infraestructura con base en datos reales.

4.3.2. Aprendizajes adquiridos

De la experiencia del trabajo realizado se considera importante destacar los aprendizajes que el mismo dejó a sus autores, considerando varios aspectos. Por un lado, la incorporación de una multitud de nuevas herramientas, cada una con sus propias bondades y todas en su conjunto favoreciendo a simplificar el trabajo diario.

Tal es así que lo aprendido se aplicó desde los primeros momentos para escribir esta tesina, que fue generada usando *LaTeX* sobre máquinas con sistemas operativos diferentes. La clave en este sentido estuvo en utilizar *Docker* para generar la versión en PDF partiendo de los fuentes. Esta herramienta incluso se compartió con el Director, Co-Director y Asesor Profesional que, sin necesidad de instalar ningún compilador de *LaTeX* en sus equipos, pudieron generar el PDF partiendo de los fuentes, haciendo uso del contenedor apropiado.

Por otro lado, la gran mayoría de los problemas aquí planteados han sido objeto de análisis en muchos momentos por parte de los autores, lo que siempre llevó a buscar soluciones que, si bien podían cumplir algunos objetivos parciales, nunca pudieron alcanzar la visión global que la metodología *DevOps* aporta, lo que incluso llevó a repensar y hacer un replanteo de la concepción de la administración de sistemas y a modificar, probablemente de forma permanente, la manera de administrar una infraestructura tecnológica.

4.4. Trabajos futuros

Del presente trabajo y de las limitaciones planteadas surgen varias temáticas con las que se puede complementar y extender lo realizado. A continuación se mencionan algunas de ellas.

4.4.1. Monitoreo

Para poder tener una visión global del estado de los servicios y los sistemas es de gran importancia contar con mecanismos de monitoreo pasivos, que sólo emitan alertas, o bien activos, pudiendo reaccionar ante diferentes situaciones. Incluso podría ser deseable la combinación de ambos. Esto es fundamental para poder actuar de manera proactiva y evitar problemas antes de que se produzcan o responder ante los mismos en el menor tiempo posible.

Ahora bien, contando con herramientas de aprovisionamiento y de automatización, los servidores pueden autoconfigurarse detectando cuáles son los equipos de monitoreo, permitiendo además que estos últimos infieran qué servicios deben verificar en cada servidor, en base a los servicios que cada servidor presta[38]. Este dinamismo garantiza que la infraestructura siempre estará monitoreada y que cada nuevo servidor que se agregue o un nuevo servicio o aplicación que se despliegue será monitoreado automáticamente.

Dentro del software para cumplir las tareas de monitoreo puede considerarse *Nagios*, *Icinga*, *Sensu*, *Monit*, entre otros.

4.4.2. Estadísticas

Sin estadísticas, cualquier decisión que se tome se realiza a ciegas, siguiendo la intuición, sobredimensionando recursos o bien dimensionando a prueba y error. Las estadísticas son de gran relevancia porque la información que brindan permite conocer en detalle muchos aspectos de la infraestructura, de los servicios, de las aplicaciones, de los recursos asignados y muchas cosas más, posibilitando la toma de decisiones informadas.

Se pueden obtener estadísticas de todo tipo, las más simples de ver podrían ser el uso de CPU, de memoria, de disco, de entrada/salida y de red en un servidor determinado. Pero también pueden obtenerse estadísticas que indiquen la cantidad de accesos que tuvo determinada aplicación, la cantidad de memoria y CPU que consumió una aplicación en particular,

[38]Chef por ejemplo cuenta con una potente herramienta de búsqueda, gestionada por el servidor de Chef que cataloga toda su infraestructura, exponiendo información a los diferentes nodos. Así, un servidor de base de datos podría por ejemplo ser etiquetado como database-server y que el servidor de monitoreo programe un chequeo que verifique por ejemplo el estado del MySQL en todos los nodos que tengan la mencionada etiqueta.

cantidad de archivos abiertos en un sistema y por aplicación, errores que tuvo cada aplicación, cantidad de conexiones recibidas en cada puerto, entre muchas opciones más.

Nuevamente, esta tarea se puede automatizar y ser parte del propio aprovisionamiento de un servidor, servicio y/o aplicación de manera de explotar el dinamismo y la flexibilidad que brindan las herramientas de aprovisionamiento.

Para obtener y visualizar estadísticas existen muchas soluciones de software, como es el caso de *Collectd*, *Graphite*, *Grafana*, *InfluxDB*.

4.4.3. Gestión de las actualizaciones

Gran parte de las herramientas que se utilizan en la *DGTIC de la Universidad Nacional de Entre Ríos* son aplicaciones web de terceros de las que se liberan nuevas versiones continuamente. Debido a que no existe ningún proceso ni política de gestión de las actualizaciones, estas quedan normalmente en manos del responsable de administrar la aplicación. Esto trae asociados dos grandes inconvenientes:

- Muchas de las actualizaciones incluyen parches de seguridad ante eventuales vulnerabilidades. El no aplicarlas supone exponer vulnerabilidades conocidas y explotables, lo que atenta contra la correcta prestación del servicio y puede poner en riesgo otros sistemas.
- Con el paso del tiempo, la no actualización de los sistemas implica que se acumulen varias versiones y que, cuando se intente actualizar, el trabajo de llevarlo a cabo sea más difícil.

La gestión de las actualizaciones es una tarea realmente compleja, debido a que no es posible conocer de antemano cuándo será liberada una nueva versión de una aplicación y esto requeriría, en todo caso, encontrar un esquema de monitoreo que ayude a detectarlo. El estudio detallado de esta problemática y una posible solución constituye un objeto de estudio complejo e interesante.

4.4.4. Infraestructura con Docker

En este trabajo se utilizó Docker de una manera sencilla, combinándolo con *Ansible* para servir algunas aplicaciones. Existen múltiples tecnologías

asociadas a Docker que llegan al punto de tener sistemas operativos dedicados a Docker como CoreOS[39] y RancherOS[40] que incluyen herramientas de descubrimiento de servicios, autoconfiguración, alta disponibilidad, entre otras. También se pueden encontrar herramientas que permiten gestionar clusters de contenedores como es el caso de Kubernetes de Google[41] o Mesos de Apache[42].

El ecosistema alrededor de Docker es complejo y dinámico por ser una tecnología nueva[43] y en pleno desarrollo. Es sin dudas una temática que debe ser considerada con especial atención debido a que la tendencia pareciera indicar que en los próximos años su uso será mucho más extendido.

4.4.5. Restauración masiva de la infraestructura

Como parte de las limitaciones de este trabajo se mencionó la tarea de restaurar la infraestructura de manera completa. En este sentido, sería ideal poder contar con una herramienta que sea capaz de realizar la automatización no ya a nivel de servidor sino a nivel centro de datos. Con dicho objetivo existen algunas tecnologías, como pueden ser:

- AWS Cloud Formation.
- Chef Provisioning.
- OpenStack Heat.
- Terraform.

Un trabajo en este sentido puede permitir, no sólo recuperación ante desastres o replicar la infraestructura para pruebas, sino que además da la posibilidad de que la misma infraestructura que actualmente la *DGTIC de*

[39] https://coreos.com

[40] http://rancher.com/rancher-os/

[41] http://kubernetes.io

[42] http://mesos.apache.org

[43] Si bien los contenedores de Linux existen hace mucho tiempo, la manera en que Docker los presenta es innovadora y les da un atractivo muy particular por la multiplicidad de herramientas con las que cuenta y por su enfoque desde el concepto de los sistemas inmutables, mencionado previamente en este trabajo.

la Universidad Nacional de Entre Ríos tiene en su propio centro de datos virtualizada con VMware pueda en un futuro migrarla a otra plataforma de virtualización, a un centro de datos externo o bien llevarla a la nube usando por ejemplo algún servicio como AWS.

4.4.6. Aprovisionamiento de la red

En el desarrollo de este trabajo se hizo foco en la automatización y el aprovisionamiento de la infraestructura de servicios, principalmente de servidores y aplicaciones. No obstante, es posible extender el alcance de la automatización planteada a todo el equipamiento de red, de manera de poder gestionar los switches, routers y firewalls también, logrando una administración global y completa e incorporando en la infraestructura de red todos los beneficios analizados.

A. ANEXO I

A.1. Tecnologías de Virtualización

La virtualización permite ejecutar, en un mismo equipo físico y por medio de software, múltiples instancias de máquinas diferentes, denominadas máquinas virtuales, cada una con su propio sistema operativo y recursos de hardware. La abstracción que provee la virtualización hace que cada máquina virtual se comporte como si fuera un equipo físico totalmente independiente.

A.1.1. VMware

VMware es la empresa líder en el mercado de virtualización[44] y una de las pioneras en esta materia[45].

Existen diferentes versiones de VMware, que incluyen algunas gratuitas pensadas para escritorio como *VMware Workstation Player* hasta robustas plataformas de virtualización como *VMware vSphere*.

A.1.2. VirtualBox

VirtualBox es una plataforma muy popular de virtualización orientada principalmente a estaciones de trabajo que suele utilizarse para tareas de desarrollo, pruebas o para disponer en una misma máquina de diferentes sistemas operativos. VirtualBox no sólo es una plataforma muy sencilla de utilizar sino que tiene además funciones avanzadas como las instantáneas y está disponible para los principales sistemas operativos como son Windows, Linux y Mac OS X. En la actualidad, VirtualBox pertenece a la empresa Oracle.

---

[44]Según lo indica Gartner en su cuadrante mágico para la virtualización de servidores x86[40]

[45]La realidad indica que el concepto de virtualización fue originado por IBM hace más de cuarenta años[15] pero, si bien el concepto es el mismo, el contexto y su aplicación se han modificado en el tiempo.

A.1.3. KVM

Kernel-based Virtual Machine[7] está formada por un módulo del núcleo de Linux que, a partir de la versión 2.6.20 del kernel, se incorporó de manera oficial al mismo. El significado de esto es que es una tecnología directamente disponible en prácticamente cualquier versión de Linux actual. Para su funcionamiento, no obstante, necesita de un procesador x86 o x86_64 que incorpore la tecnología de virtualización y la tenga habilitada. Esta característica también existe en cualquier máquina actual, aunque no suele encontrarse habilitada por defecto.

Hoy en día, KVM pertenece a la empresa Red Hat.

A.2. Vagrant

Es una herramienta que permite gestionar máquinas virtuales de forma rápida y sencilla a través de la línea de comandos. Es muy práctica para ser usada en estaciones de trabajo para tareas de desarrollo y pruebas. Originalmente pensada para ser utilizada con la plataforma de virtualización[17] VirtualBox, actualmente soporta otros proveedores como VMware o AWS[46].

Vagrant permite agilizar y automatizar tareas a través de su sencilla e intuitiva línea de comandos y de archivos de configuración autoexplicativos. Sin entrar en aspectos técnicos detallados, para los cuáles se puede recurrir a la documentación de Vagrant[47], se explica brevemente a continuación el uso de la herramienta para mostrar su potencia.

Para inicializar una instalación de Vagrant se utiliza el comando `vagrant init`, al que se le puede especificar una versión de la máquina que se desea utilizar, como puede verse debajo.

```
$ vagrant init hashicorp/precise32

  A ‘Vagrantfile‘ has been placed in this directory. You are now
  ready to ‘vagrant up‘ your first virtual environment! Please read
  the comments in the Vagrantfile as well as documentation on
```

[46] Amazon Web Services es un conjunto de servicios computacionales en la nube que tiene entre sus servicios a EC2, Amazon Elastic Compute Cloud, que brinda principalmente la posibilidad de disponer de servidores virtuales en la nube.

[47] https://docs.vagrantup.com/v2/

```
‘vagrantup.com‘ for more information on using Vagrant.
```

Así, el comando anterior creará un archivo de configuración de *Vagrant* llamado *Vagrantfile* con todas las directivas de configuración de la máquina virtual. Luego, al ejecutar el comando `vagrant up` se creará la máquina virtual en cuestión utilizando *VirtualBox* con el sistema operativo *Ubuntu 12.04 LTS 32-bit*, que es el correspondiente a la imagen indicada con el nombre `hashicorp/precise32`.

```
$ vagrant up
```

```
Bringing machine 'default' up with 'virtualbox' provider\ldots
```

Cuando el comando `vagrant up` finalice con éxito la máquina ya estará instalada y disponible para ser utilizada. Así, es posible loguearse a la misma a través de SSH con el siguiente comando:

```
$ vagrant ssh
```

Cuando se termine de trabajar con la máquina, la misma puede apagarse haciendo uso del comando `vagrant halt` y destruirse con `vagrant destroy`, como se muestra debajo:

```
$ vagrant destroy
```

```
default: Are you sure you want to destroy the 'default' VM? [y/N]
```

Con el comando anterior, la máquina se elimina directamente del sistema.

Puede verse, con la breve introducción dada, la potencia de la herramienta y su sencillez para gestionar las máquinas virtuales.

A.2.1. Aprovisionamiento

Vagrant soporta utilizar herramientas de aprovisionamiento para instalar y configurar cada máquina que pueden ejecutarse al invocar el comando `vagrant up` o bien `vagrant provision`. Dicho aprovisionamiento puede realizarse utilizando un simple *shell script* o utilizando incluso herramientas más sofisticadas como *Ansible*, *Chef*, *Puppet* y *Salt*, las cuales se tratan en otro apartado.

A.2.2. Generadores de *Vagrantfile*

Existen varios sitios web que permiten generar los archivos *Vagrantfile* por medio de una interfaz visual, de manera intuitiva y mediante unos pocos clicks. Algunos de dichos sitios son:

- puphpet.com
- phansible.com
- rove.io

A.3. Contenedores

La tecnología de contenedores de Linux[13] [31] se diferencia en muchos aspectos de la virtualización completa, y tiene muchas ventajas respecto de esta, con algunas desventajas también, aunque su aplicación es más limitada que la de la virtualización debido a que tiene ciertas restricciones fuertes que hacen que no siempre sea posible su uso.

En líneas generales, se puede caracterizar a los contenedores considerando los siguientes aspectos:

- Sólo pueden ser ejecutados sobre servidores cuyo sistema operativo sea Linux y tenga soporte para esta tecnología en el kernel.
- Comparten el kernel con el *sistema operativo host*.
- Utilizan tecnologías como *cgroups*[38] [30] y *namespaces*[39] para el manejo de recursos y el aislamiento de los procesos.

Comparando los contenedores con la virtualización, se pueden mencionar las siguientes ventajas y desventajas de cada uno:

Virtualización

- Permite ejecutar cualquier sistema operativo, sin importar incluso cuál sea el *sistema operativo host*.
- Crea un entorno virtual completo, dando como resultado una separación real del *sistema operativo host*, lo que aisla los problemas que puedan existir entre contenedores y da mayor seguridad a cada entorno virtual.

- Al ser entornos virtuales completos, las aplicaciones funcionan sin necesidad de ningún cambio ni adaptación específicos.

Contenedores

- Son extremadamente livianos, lo que permite desplegar en un mismo *sistema operativo host* una cantidad mucho mayor de contenedores que la cantidad de máquinas virtuales que pueden crearse.
- Prácticamente no tienen ningún *overhead*, lo que los hace muy rápidos y permite que inicien en muy pocos segundos.
- Al compartir varios de los recursos con el *sistema operativo host*, suelen ocupar menos espacio en disco que las máquinas virtuales.

Es importante destacar que ambas tecnologías no son mutuamente excluyentes sino que pueden convivir y ser utilizadas en conjunto[18]. De hecho, en la actualidad lo más probable es que cualquier contenedor se ejecute en realidad sobre un servidor virtual y no sobre un servidor físico.

A continuación se hará un repaso por las diferentes opciones que existen para crear contenedores de Linux.

A.3.1. chroot

El término *chroot*, que significa literalmente "change root directory", es decir, cambiar el directorio raíz (/), sirve para crear entornos virtuales en un sistema operativo *Unix*, generando de alguna manera una separación entre la estructura de directorios del sistema operativo y la del entorno virtualizado. Así, un programa que se ejecute utilizando *chroot* verá como directorio raíz un directorio que es, en realidad, algún subdirectorio del *sistema operativo host*.

La estrategia de *chroot* es usada con diferentes propósitos: por un lado, se configuran algunos procesos en un entorno de este tipo para mejorar la seguridad de los mismos, por ejemplo en el caso de servidores FTP, donde cada cliente al conectarse lo hace con un *chroot* en su directorio de usuario. Así se evita que ese usuario pueda subir en la jerarquía de directorios accediendo a otras ubicaciones que no corresponden.

Otro uso muy común de *chroot* es por ejemplo para rescatar un sistema operativo que no pueda iniciarse correctamente. En estas situaciones, utilizando un *Live CD* u otro sistema funcionando se inicia un sistema *Unix*, se

monta la partición raíz del sistema a rescatar en algún subdirectorio y se hace un *chroot* en dicho directorio. De esta manera se trabaja en el sistema operativo a rescatar como si hubiera iniciado normalmente.

Por la naturaleza de *chroot*, que confina a un proceso a un subdirectorio del *sistema operativo host* del que el proceso no puede salir, se lo suele denominar *jail* (jaula).

Se puede considerar que en cierto modo la técnica de *chroot* es un antecedente de los contenedores de Linux aunque, a diferencia de estos, *chroot* no provee aislamiento de procesos sino simplemente de directorios.

A.3.2. OpenVZ

Es una tecnología de contenedores que permite correr múltiples instancias de sistemas operativos aislados, usa un agregado a Linux y solamente puede correr sistemas Linux[12].

Cada contenedor es una identidad separada y se la puede ver como una máquina real porque maneja sus propios archivos, usuarios, procesos y dirección de red. Soporta migración en vivo, que consiste en mover un contenedor de un servidor a otro sin apagar en ningún momento el contenedor que está en ejecución.

Una ventaja muy atractiva de este tipo de contenedores es que soportan la asignación dinámica de procesadores, memoria y disco sin necesidad de apagar el contenedor, pudiendo redimensionarlo en cualquier momento sin ninguna interrupción del servicio prestado.

A.3.3. Linux Containers: LXC

LXC es una interfaz que puede ejecutarse en espacio de usuario y permite administrar contendedores de forma sencilla[8]. Alcanzó su versión estable en Febrero de 2014 y está integrada de forma nativa en el Kernel de Linux. Tiene herramientas para gestionar los contenedores por línea de comandos y también a través de una API que, aunque se anuncia como potente, aún resulta difícil encontrar documentación clara sobre la misma, aunque los fuentes de LXC pueden hallarse fácilmente en su repositorio de Github[48].

Actualmente, la tendencia apunta a que *LXC* reemplace eventualmente a

[48]https://github.com/lxc/lxc

OpenVZ que además nunca terminó de ser integrado en el Kernel de Linux[49]. Esta discusión hoy en día puede ser muy subjetiva, debido a que no sólo *LXC* es una tecnología muy reciente sino que basa su propio funcionamiento en otras tecnologías también muy nuevas, lo que aún sigue siendo argumento a favor de *OpenVZ* que es una tecnología robusta y vigente desde hace bastante más tiempo.

No obstante, incluso plataformas como *Proxmox*[50] que ha incluido *OpenVZ* durante años como una de sus principales tecnologías de "virtualización" y ha sido una de las grandes responsables de su popularización, ha tomado la decisión de reemplazar *OpenVZ* con *LXC*[51].

A continuación se muestra, a través de un simple ejemplo[9] la operatoria básica de *LXC*, creando un contenedor y haciendo uso del mismo.

```
# lxc-create -t download -n my-container
```

Con el comando anterior se lista una serie de posibles imágenes para descargar, se escoge Debian Wheezy de 32 bits.

```
---
DIST    RELEASE ARCH    VARIANT          BUILD
---
centos  6       i386    default   20150319_02:16
debian  jessie  i386    default   20150319_22:42
debian  sid     i386    default   20150319_22:42
debian  wheezy  i386    default   20150319_22:42
fedora  19      i386    default   20150319_01:27
fedora  20      i386    default   20150319_01:27
gentoo  current i386    default   20150319_14:12
oracle  6.5     i386    default   20150319_11:40
plamo   5.x     i386    default   20150319_21:36
ubuntu  precise i386    default   20150318_03:49
ubuntu  trusty  i386    default   20150319_03:49
ubuntu  utopic  i386    default   20150319_03:49
ubuntu  vivid   i386    default   20150319_03:49
---
```

[49]Este tipo de contenedores necesitan de un Kernel especial para ejecutarse.

[50]https://www.proxmox.com

[51]https://www.proxmox.com/en/news/press-releases/proxmox-ve-4-0-released

```
Distribution: debian
Release: wheezy
Architecture: i386

Downloading the image index
Downloading the rootfs
Downloading the metadata
The image cache is now ready
Unpacking the rootfs

---
You just created a Debian container

To enable sshd, run: apt-get install openssh-server

For security reason, container images ship without user
accounts and without a root password.

Use lxc-attach or chroot directly into the rootfs to set
a root password or create user accounts.
```

Al finalizar la ejecución anterior se cuenta con el contenedor ya creado. Puede verse en el mensaje que se lee una vez terminada la misma que dice explícitamente que por cuestiones de seguridad el contenedor no cuenta con ningún servicio remoto de administración y que sólo es posible acceder al mismo desde el *sistema operativo host* a través de una utilidad específica.

Este contenedor se creó sobre un sistema operativo Ubuntu 14.04, como puede verse debajo:

```
# cat /etc/issue

Ubuntu 14.04.2 LTS \n \l
```

Teniendo lo anterior en cuenta, con los siguientes dos comandos se inicia el contenedor y se establece una conexión con el mismo.

```
# lxc-start -n my-container
# lxc-attach -n my-container
```

Para verificar que el entorno es efectivamente diferente del sistema operativo real puede verse que, como se espera, el contenedor ejecuta un sistema operativo Debian 7.

```
root@my-container:~# cat /etc/issue

Debian GNU/Linux 7 \n \l
```

A.3.4. Docker

Docker es una plataforma de contenedores de código abierto[52] que se basa en *LXC* pero que aporta un conjunto enorme de herramientas de alto nivel sobre la mencionada tecnología, que es de más bajo nivel[27]. Así, Docker provee una serie de características a destacar:

- Es portable, con lo que un contenedor Docker puede ser ejecutado en cualquier máquina con soporte para dicha tecnología.
- Está particularmente optimizado y enfocado al despliegue de aplicaciones, a diferencia de otras tecnologías de contenedores que buscan generar un servidor virtual con las funciones completas de cualquier equipo.
- Su construcción es automatizada, desde un código fuente se puede generar un nuevo contenedor de la forma exactamente definida.
- Todo contenedor Docker es versionado, esto quiere decir que se puede acceder a la historia de cada cambio en un contenedor de Linux como si se pudiera repasar cada cambio, por más simple que sea, en un servidor virtual.
- Reusabilidad, debido a que todo contenedor puede ser base de otro nuevo contenedor, pudiendo agregársele funcionalidades sucesivas y de forma incremental.
- Posibilidad de compartirse con otras personas, a través del registro público de Docker, denominado *Docker Hub*.
- Cuenta con un completo ecosistema de herramientas, desde herramientas nativas de Docker y su propia API, hasta múltiples utilidades y tecnologías que comenzaron a desarrollarse alrededor de Docker.

[52] https://github.com/docker/docker

Existen múltiples herramientas, al punto que se han desarrollado sencillas utilidades por línea de comandos e incluso con interfaces gráficas que permiten ejecutar, haciendo uso de una máquina virtual apropiada, contenedores Docker en sistemas operativos como Windows o Mac OS X, una integración que no existe con otro tipo de contenedores.

Para comprender mejor el funcionamiento de Docker se describen tres componentes esenciales.

A.3.4.1 Imágenes

Una *Docker image* es un *template* de sólo lectura. Por ejemplo, una imagen puede ser un *Ubuntu* con un servidor web Apache y una aplicación ya instalada. Luego, son estas imágenes las que se utilizan para crear los contenedores.

A.3.4.2 Registros

Un *Docker registry* es un repositorio público o privado desde dónde se pueden subir o bajar las imágenes. Hay en particular un registro público llamado *Docker Hub*[53] que es oficial de Docker y provee una gran colección de imágenes, desarrolladas por fuentes verificadas y también por terceros que usan el repositorio para publicar sus propias imágenes de Docker y compartirlas con otras personas..

Docker no es solo un producto, es un patrón de diseño[21] que reutiliza tecnologías ya existentes.

A.3.4.3 Contenedores

Los *Docker containers* se pueden ver como un directorio que contiene todo lo que se necesita para que la aplicación corra. Cada contenedor es creado a partir de una imagen. Los contenedores pueden arrancarse, pararse, moverse y borrarse.

[53]https://hub.docker.com/

A.3.4.4 Cómo funciona Docker

Cada imagen consiste en una serie de capas, las cuales son combinadas utilizando *Union File Systems*[16] que permite que archivos y directorios de sistemas de archivos distintos, conocidos como ramas, se superpongan de forma transparente para formar un único sistema de archivos.

Estas capas hacen que cuando se actualiza una aplicación en una imagen, una nueva capa se construye en la actualización, y solamente esta capa es la que se sube. De esta manera, como no se necesita distribuir la imagen completa sino solamente esta última capa de diferencia, las actualizaciones de un contenedor suelen ser bastante rápidas.

Los contenedores son construidos a partir de las imágenes, recordar que estas son de sólo lectura con lo cual se agrega una capa adicional de lectura/escritura por encima de la imagen.

Ejemplo, si se quiere arrancar un nuevo contenedor a partir de la imagen *Alpine*[54] :

```
$ docker run -t -i alpine /bin/sh
/ # cat /etc/issue
Welcome to Alpine Linux 3.1
/ # rm -rf /
/ # ls
/bin/sh: ls: not found
```

Con el comando rm -rf / se destruye todo el sistema del contenedor. Pero como se menciona anteriormente, las imágenes son de sólo lectura y los contenedores son instancias de las mismas, agregando una capa de lectura/escritura con lo cual cualquier cambio, para bien o para mal, quedará en esta capa.Siguiendo con el ejemplo al volver ejecutar:

```
$ docker run -t -i alpine /bin/sh
/ # ls
bin dev etc home lib mnt proc root run sbin sys tmp usr var
```

Se obtiene un nuevo contenedor, totalmente funcional que se crea a partir de la imagen *Alpine*.

[54] *Alpine Linux* es una distribución muy liviana orientada a la seguridad. http://www.alpinelinux.org/

Estas operaciones demoran fracciones de segundos en una computadora personal, ventaja interesante respecto de cualquier plataforma de virtualización que requiere clonar un disco virtual, algo que, por el tamaño que suelen tener los archivos de los discos virtuales, puede llevar varios minutos.

Notar que en el caso de que la imagen *Alpine* no se encuentre localmente, *Docker* hace lo que se denomina un *pull* de la imagen, es decir descarga todas las capas necesarias con las cuales es posible construir la imagen *Alpine*.

En las ejecuciones anteriores se puede observar que *Docker* maneja otro concepto interesante, que es ejecutar un sólo proceso como un microservicio, esto quiere decir que el contenedor no pasa por un proceso de inicio completo como cuándo se utilizan *SysV init* o *Upstart*.

A.3.4.5 Dockerfile

Un *Dockerfile* es un archivo de texto que tiene todos los comandos que deberían ejecutarse normalmente de forma manual para poder construir una *Docker image*[3].

Con el siguiente comando, por ejemplo, se puede construir una imagen de Docker.

```
docker build
```

Como ejemplo se puede mencionar la creación de un contenedor para compilar *LaTeX*. La instalación de *LaTeX* puede llegar a ser compleja, además de requerir muchas utilidades solamente para poder generar un archivo *PDF*. Con la idea de los contenedores, se puede construir uno que permita precisamente compilar archivos *LaTeX*. De esta manera, se obtiene un compilador de *LaTeX* portable, al que se le pueden agregar nuevas características reconstruyendo el contenedor por medio de un archivo *Dockerfile*.

A continuación se muestra un ejemplo de un archivo *Dockerfile* que crea el contenedor para realizar la compilación de *LaTeX*.

```
FROM ubuntu:trusty
MAINTAINER Joaquin Salvarredy <joaquin@salvarredy.com.ar>
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update -q
RUN apt-get install -qy texlive-latex-base texlive-lang-spanish \
                        texlive-binaries texlive-latex-extra \
```

```
                        xindy python-pygments gnuplot

RUN apt-get remove -y texlive-latex-base-doc \
                      texlive-latex-extra-doc

RUN apt-get -y autoremove && \
    apt-get -y clean && \
    rm -rf /var/lib/apt/lists/* && \
    rm -rf /tmp/*

WORKDIR /data
VOLUME ["/data"]
```

Para construir la imagen:

```
$ docker build -t jsalvarredy/latex .
```

Para generar el *PDF* final, ejecutar:

```
$ docker run --rm -i -v $PWD:/data jsalvarredy/latex \
    pdflatex -shell-scape main
```

El *Dockerfile* probado y funcionando, se sube a *GitHub*[55], y se configura el registro público de imágenes *Docker hub*, para que, a partir de la información existente en el repositorio de *Github*, construya de forma automática la correspondiente imagen de *Docker*. Así, ya no se necesita hacer la reconstrucción de la imagen, solamente ejecutando la última línea se puede compilar archivos *LaTeX* desde cualquier plataforma y lugar con acceso a Internet.

[55]GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones GIT. Ver https://github.com/

B. ANEXO II

B.1. Ansible

Es una herramienta para configurar y administrar sistemas que permite gestionar múltiples nodos[1]. No necesita ningún tipo de agente especial en los equipos remotos dado que basa su funcionamiento sobre el uso del protocolo *ssh*[22], lo cual es una gran ventaja si se lo compara con otras herramientas similares. Con Ansible no es necesario disponer de un servidor central, lo que simplifica mucho su uso y no genera infraestructura adicional para utilizarlo. Al no requerir agentes especiales en los sistemas remotos, la gestión de cualquier equipo se simplifica y agiliza. Esta simplicidad sacrifica, no obstante, varias funciones avanzadas, presentes en otras tecnologías similares, que hacen de Ansible una herramienta fácil de aprender y utilizar pero con un potencial menor que sistemas como Chef[56] y Puppet[57].

B.1.1. Playbooks

Es el término que utiliza *Ansible* para denominar al lenguaje por medio del cuál describe la configuración, el despliegue y el aprovisionamiento a realizar sobre los sistemas correspondientes. Los *Playbook* se escriben en archivos de texto plano, respetando el formato *YAML*, en donde se describen las políticas aplicar a los sistemas remotos.

Un ejemplo de la estructura de un *Playbook*:

```
1  - hosts: db
2    vars:
3        pgversion: 8.4
4    tasks:
5      - name: Se cambia los repositorios por uno local
6        copy: src=data/sources.list dest=/etc/apt/sources.list
7
8      - name: Se actualizan los paquetes
9        shell: apt-get update
10
```

[56]Ver la sección [B.3] en la página 95 para más información sobre Chef.

[57]Ver la sección [B.2] en la página 93 para más información sobre Puppet.

```
11      - name: Se instalan utilidades como rsync, screen, mc
12        action: apt pkg={{item}} state=present
13        with_items:
14         - rsync
15         - screen
16         - mc
17
18      - name: Se instala el servidor de BD postgres
19        action: apt pkg={{item}} state=installed
20        with_items:
21         - postgresql
22         - postgresql-{{ pgversion }}
23         - postgresql-client-{{ pgversion }}
24
25      - name: Se reinicia el servidor para que tome todos los cambios
26        command: /sbin/reboot
```

El poder expresivo que tiene la sintaxis *YAML* hace que en pocas líneas se pueda lograr una instalación bien documentada. Es posible parametrizar los *Playbook* utilizando variables, subir archivos al sistema que se está instalando, ejecutar cualquier tipo de comando, y hasta incluso es posible reiniciar los equipos remotos una vez instalado todo el sistema si alguna modificación así lo exigiera.

Una de las ventajas de utilizar herramientas cuya configuración se exprese a través de archivos de texto, es que las soluciones obtenidas son fácilmente versionables, lo que permite mantener una historia de las mismas y compartirlas con otros administradores.

B.1.2. Hosts

Por cada ejecución de un *Playbook*, se deben elegir cuales son las máquinas remotas a las que se le aplican los pasos definidos en el mismo. Los *Hosts* pueden ser equipos individuales o un grupo de sistemas. En el ejemplo anterior, la línea `hosts: db` indica que el *Playbook* se debe aplicar al *host* llamado *db*.

B.1.3. Lista de *Tasks*

En los *Playbook* se definen una lista de *Tasks*(Tareas), que son ejecutadas en orden y una a la vez. Si la ejecución es sobre un grupo de *hosts*, *Ansible*

se asegura que cada *Playbook* finalice en todos los nodos antes de avanzar a la siguiente tarea.

Si una tarea falla en un *host*, se lo saca de la rotación para todo lo que reste del *Playbook*, con lo cual se debe corregir el archivo del *Playbook* y correr nuevamente.

El objetivo de cada tarea es ejecutar un módulo con argumentos específicos. Estos módulos son idempotentes, lo que quiere decir que se pueden aplicar infinitas veces que el resultado, mientras que el módulo no cambie, será siempre el mismo. Incluso, sólo se registra efectúa algún cambio cuando realmente se desea modificar algo en el equipo.

Cada tarea debe tener un nombre, el cual se incluye en la salida de la ejecución de una receta. Como esta salida es para lectura de un operador de sistemas, es conveniente tener buenas descripciones para ir viendo el progreso de cada tarea.

B.1.4. Roles

Mientras que es posible escribir un *Playbook* en un solo archivo de gran tamaño, al igual que en cualquier lenguaje de programación es más conveniente reusar código entre diferentes *Playbook*, que utilizar *copy/paste*, ya que además no es una buena práctica de programación. En *Ansible* se puede reusar un conjunto de tareas agrupándolas en lo que se denominan *Roles*.

La idea de estos *Roles* es permitir que se puedan definir ciertos comportamientos de un servidor, así se comienza en pensar por ejemplo en hosts que son webservers, o en hosts que son dbservers, dando un nivel de reusabilidad y abstracción.

B.2. Puppet

Puppet es una herramienta de código abierto escrita en Ruby y cuyo objetivo es la automatización y el aprovisionamiento de una infraestructura de TI. Utiliza para describir la configuración de los sistemas un lenguaje declarativo de fácil lectura llamado *Puppet Domain Specific Language (DSL)*. Este lenguaje permite, de una manera simple y concisa, describir el estado deseado de cada una de las máquinas de la infraestructura. Y es *Puppet* el responsable de utilizar las descripciones provistas para instalar las máquinas exactamente como dichas descripciones lo indican.

La sintaxis de *DSL* da independencia del sistema operativo al momento de definir qué paquetes hay que instalar, qué servicio se necesita ejecutar, cuáles son las cuentas de usuarios necesarias, cuáles los permisos otorgados, etc.

B.2.1. Arquitectura Maestro y Agentes

Puppet tiene una arquitectura maestro/agente, en donde el servidor maestro *Puppet* es el responsable de manejar y configurar los nodos agentes en base a sus requerimientos. Los nodos que son manejados corren la aplicación *Puppet agent* como un servicio, y uno o más servidores corren la aplicación *Puppet master*, usualmente como una aplicación *Rack*[58] manejada por un servidor web.

Periódicamente, los agentes envían peticiones al maestro solicitando un catálogo. El maestro, utilizando distintas fuentes de información, compila y retorna a los clientes el catálogo solicitado.

Una vez recibido el catálogo, los agentes deben aplicar chequeando cada recurso según lo que describe el catálogo. Si alguno de los agentes detecta que en su sistema alguno de los recursos no está en el estado deseado, debe hacer los cambios para dejarlo en el estado correcto. Luego de la ejecución, cada agente le envía al maestro un reporte de los cambios.

B.2.2. Puppet Enterprise (PE)

Es una plataforma completa de manejo de configuración con un conjunto de componentes optimizados. Se combina la versión de código abierto *Puppet* con MCollective, PuppetDB, Hiera y más de 40 proyectos de código abierto que Puppet Labs distribuye, certifica y optimiza para lograr una solución de automatización pensada para infraestructuras críticas. Además, incluye una interfaz web para analizar los reportes y controlar la infraestructura.

[58]Rack provee una interfaz modular y adaptable para aplicaciones web desarrolladas en Ruby.

B.2.3. Módulos

Puppet está pensado para trabajar con módulos que hacen una tarea específica. Así, existen módulos para el manejo de usuarios y módulos para la configuración de un servidor de hora. Existen, como puede suponerse, diferentes módulos, cada uno diseñado para una tarea en particular. Esto fomenta la reutilización de código. Los módulos se colocan en un repositorio público para que de esta forma la comunidad pueda contribuir aportando sus propios módulos, además de los muchos que mantienen las personas que forman Puppet Labs.

El mecanismo de instalación de los módulos es muy sencillo. Si, por ejemplo, se quiere instalar el módulo Apache que provee todo lo necesario para la instalación automática del servidor web Apache, se debe escribir el siguiente comando:

```
# puppet module install puppetlabs-apache
```

Este comando baja el módulo del repositorio público y lo instala en el lugar donde están los módulos para que pueda ser utilizado por el Puppet Master.

B.3. Chef

Chef es a la vez un framework de desarrollo que permite, a través de código Ruby y varias DSL, programar de forma completa cualquier infraestructura de TI, desde su definición inicial hasta los tests para la misma y, por otro lado, cuenta con un importante conjunto de herramientas de gestión, no sólo del código sino también de los servidores. A través de diversos plugins, es posible con Chef crear de forma dinámica e incluso automática servidores en plataformas VMware, Proxmox e incluso en proveedores en la nube como Amazon, pudiendo indicar al momento de la creación qué se espera que el servidor tenga instalado y configurado cuando se cree.

Chef es de código abierto y comparable con Puppet por dimensión y complejidad. Si bien su arquitectura, en líneas generales, es bastante simple, existen muchos conceptos asociados al producto que son necesarios conocer. Debajo, se irán describiendo cada uno de ellos con el nombre en español y el correspondiente en inglés entre paréntesis.

- **Receta (recipe):** una receta, como en la cocina, es una serie de pasos y/o acciones que deben llevarse a cabo para obtener un resultado espe-

rado. En Chef, una receta es la unidad básica de desarrollo. Un ejemplo sería una receta que instale Nginx.

- **Libro de cocina (cookbook):** Chef organiza todas sus recetas en libros de cocina, que reúnen recetas relacionadas. Siguiendo con el ejemplo anterior, un libro de cocina podría contener la receta que instale Nginx, que lo configura y que crea los hosts virtuales. No existen reglas al respecto que indiquen cómo delimitar un libro de cocina en Chef y queda más bien en manos del desarrollador de qué forma prefiere estructurar su desarrollo. No obstante, como en todo framework de desarrollo y con la experiencia de uso, comienzan a surgir buenas prácticas y patrones de desarrollo que han probado ser más convenientes por diferentes motivos.

- **Ambiente (environment):** un ambiente en Chef tiene el mismo significado que en cualquier entorno de TI y representa un conjunto de servidores que tienen determinado "estado" como conjunto, como pueden ser los ambientes de producción, desarrollo y testing.

- **Nodo (node):** un nodo en Chef es cualquier máquina, sea servidor o de escritorio, física o virtual, que haya sido aprovisionada y sea gestionada con Chef. Un nodo atraviesa, la primera vez, un proceso de configuración inicial o bootstrap en el que se instala en el mismo un software agente (chef-client) y se establece el mecanismo de comunicación con el servidor de Chef. Luego de ello el nodo, en base a las recetas que tenga aplicadas y a los atributos que contenga, se configurará según lo que las propias recetas indiquen.

- **Bolsa de datos (data bag):** una bolsa de datos es una estructura de datos, representada normalmente en formato JSON, que contiene elementos de la forma clave-valor y que sirve para definir configuraciones y datos necesarios para el proceso de aprovisionamiento y configuración de los servidores. En Chef, los data bags pueden ser planos o encriptados, para los casos donde los mismos reúnan información sensible.

- **Lista de ejecución (run-list):** una lista de ejecución se compone de roles y/o recetas que se ejecutan en el mismo orden en que se indican y es propia de cada nodo, ya que los nodos pueden tener la misma lista de ejecución o tener una propia. En la mayoría de los casos, la lista de ejecución variará entre nodos, debido a que es a través de ella que los mismos instalan el software necesario para cumplir su objetivo y se configuran en pos de alcanzarlo. Algo importante de la lista de

ejecución es que si una receta o rol aparece en ella más de una vez, sólo será ejecutado una sola de esas veces (la primera).

- **Rol (role):** un rol permite identificar un grupo de nodos que tienen una función específica y común a todos y que necesitan y comparten atributos y una lista de ejecución. De esta forma, se puede tener un rol que sea "monitoring_server", que tenga en su lista de ejecución la receta para instalar Nagios y algunos atributos específicos para esa receta. Así, se puede aplicar ese rol a todos los servidores de monitoreo de la organización, lo que agiliza el procedimiento y asegura la consistencia entre los equipos.

- **Servidor de chef (chef server):** tiene tres funciones elementales y muy importantes a su vez:

  - Actúa como un repositorio centralizado de la información para los servidores y los administradores. Aquí se guardan los libros de cocina, las recetas, la información de los nodos, las bolsas de datos, los ambientes, etcétera.
  - Como es precisamente el repositorio de todos los datos de la infraestructura, los servidores toman de él toda la información que necesitan para configurarse y lo hacen de manera periódica, cada vez que se ejecuta en ellos el proceso del software agente (chef-client).
  - Es el intermediario entre el administrador y los servidores, debido a que la interacción entre el primero con los segundos se realiza a través de los datos que este provee.

- **Cuchillo (knife):** es una herramienta que se utiliza por línea de comandos y permite interactuar con el servidor de Chef, para crear, eliminar y modificar nodos, ambientes, libros de cocina, etcétera, a la vez que también brinda la posibilidad de realizar búsquedas y ejecutar comandos sobre toda la infraestructura.

- **Cliente de chef (chef-client):** es el agente de software que se instala en todos los servidores que se gestionan con Chef y es, por medio de este, que se realiza la comunicación de los nodos con el servidor de Chef y el responsable de descargar toda la información necesaria para ejecutar la instalación y las actualizaciones.

- **Estación de trabajo (workstation):** es normalmente la máquina desde dónde los usuarios hacen todo el trabajo, que consistirá en desarrollar nuevos cookbooks, usar knife, mantener el versionado del código

en el repositorio correspondiente y sincronizarlo con el servidor de Chef, interactuar con los nodos, entre otras.

- **Ohai:** es una herramienta que detecta atributos en un nodo y los pone a disposición del cliente de chef para que puedan ser usados en cada ejecución. Entre las cosas que detecta ohai se encuentra la plataforma, información de hardware como CPU, memoria, disco, uso de red, kernel, nombre del equipo, etcétera. Es una herramienta muy útil no sólo para tener información real de los nodos aprovisionados con Chef sino también para el desarrollo de cookbooks, ya que permite programar recetas de forma más genérica.

B.4. SaltStack

SaltStack es una herramienta de software de código abierto y desarrollada en Python que permite gestionar la infraestructura[14] utilizando mecanismos de orquestación, automatización y aprovisionamiento.

A continuación se describen, de forma muy breve, los componentes generales de SaltStack.

B.4.1. Componentes

- **Salt Master:** es quién envía comandos y configuraciones a los Salt Minion que están corriendo sobre el sistema gestionado.
- **Salt Minion:** son los que reciben comandos y configuraciones desde el Salt Master.
- **Execution Modules:** comandos ejecutados bajo demanda desde la línea de comandos contra uno o más de los sistemas manejados, útiles para monitoreo en tiempo real, ver el estado del sistema o para actualizar algo critico.
- **Formulas (States):** una representación declarativa o imperativa de la configuración de un sistema.
- **Grains:** información estática que se carga al arrancar un Salt Minion, como el kernel que está corriendo o el sistema operativo.
- **Pillar:** variables definidas por el usuario que se almacenan en el Salt Master y son asignadas a uno o más minion. Los datos almacenados

pueden ser puertos, path de archivos, parámetros de configuración o passwords.

- **Top File:** combina Pillar y Formulas que se aplican a los Salt Minions.
- **Runners:** módulos que se ejecutan en el Salt Master, reportan estado de los trabajos, estado de la conexión, leen datos de APIs externas, consulta que Salt Minions están conectados.
- **Returners:** envían datos desde los Salt Minion a otro sistema, por ejemplo una base de datos.
- **Reactors:** disparan un trigger cuando ocurre un evento en el ambiente.
- **Salt Cloud / Salt Virt:** aprovisiona sistema en proveedores de la nube y los pone bajo el manejo del ambiente.
- **Salt SSH:** ejecuta comandos Salt sobre sistemas que no tienen un Salt Minion.

Referencias

[1] Ansible documentation — ansible documentation. http://docs.ansible.com/. (Visitado el 29/05/2015).

[2] Containercon interview with jerome petazzoni from docker — opensource.com. https://opensource.com/business/15/8/interview-jerome-petazzoni-docker-linuxcon?mkt_tok=3RkMMJWWfF9wsRonuqTMZKXonjHpfsX57e0tWKGzlMI%2F0ER3fOvrPUfGjI4ATsZnI%2BSLDwEYGJlv6SgFQ7LMMaZq1rgMXBk%3D. (Visitado el 12/08/2015).

[3] Docker documentation. https://docs.docker.com/. (Visitado el 29/05/2015).

[4] Docker wordpress tutorial - docker. http://www.dockerwordpress.com/. (Visitado el 10/10/2015).

[5] How to use docker to optimise a wordpress website hosted on a vps - ovh. https://www.ovh.co.uk/g1708.how_to_use_docker_to_optimise_a_wordpress_website_hosted_on_a_vps. (Visitado el 10/10/2015).

[6] jwilder/nginx-proxy · github. https://github.com/jwilder/nginx-proxy. (Visitado el 10/10/2015).

[7] Kernel-based virtual machine - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine. (Visitado el 29/05/2015).

[8] Linux containers. https://linuxcontainers.org/. (Visitado el 29/05/2015).

[9] Lxc 1.0: Your first ubuntu container [1/10] — stéphane graber's website. https://www.stgraber.org/2013/12/20/lxc-1-0-your-first-ubuntu-container/. (Visitado el 29/05/2015).

[10] Manifiesto por el desarrollo Ágil de software. http://agilemanifesto.org/iso/es/. (Visitado el 17/06/2015).

[11] Nginx reverse proxy — nginx. https://www.nginx.com/resources/admin-guide/reverse-proxy/. (Visitado el 10/10/2015).

[12] Openvz - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/OpenVZ. (Visitado el 29/05/2015).

[13] Operating-system-level virtualization - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Operating-system-level_virtualization. (Visitado el 29/05/2015).

[14] Saltstack. http://docs.saltstack.com/en/latest/. (Visitado el 29/05/2015).

[15] Soluciones de virtualización. http://www.ibm.com/midmarket/mx/es/virtualizacion.html. (Visitado el 25/10/2015).

[16] Unionfs - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/UnionFS. (Visitado el 20/10/2015).

[17] Vagrant documentation. http://docs.vagrantup.com/v2. (Visitado el 29/05/2015).

[18] Virtualization vs. containerization —. http://mainframedebate.com/2015/03/20/virtualization-vs-containerization/. (Visitado el 29/05/2015).

[19] John Allspaw and Paul Hammond. 10+ deploys per day. https://www.youtube.com/watch?v=LdOe18KhtT4, Junio 2009. (Visitado el 01/06/2015).

[20] David J. Anderson. *Kanban*. Blue Hole Press Inc, Noviembre 2013.

[21] Dan Bartow. How docker has made life awesome for soasta devops. http://www.soasta.com/blog/docker-awesome-devops/?mkt_tok=3RkMMJWWfF9wsRonuqTMZKXonjHpfsX57e0tWKGzlMI%2F0ER3fOvrPUfGjI4ATcdqI%2BSLDwEYGJlv6SgFQ7LMMaZq1rgMXBk%3D. (Visitado el 15/07/2015).

[22] Jeff Gonzalez Brandon Keown. Ansible orchestration - youtube. https://www.youtube.com/watch?v=47fKY6hrbMg, Septiembre 2013. (Visitado el 30/05/2015).

[23] Ben Butler-Cole. Rethinking building on the cloud: Part 4: Immutable servers. https://www.thoughtworks.com/insights/blog/rethinking-building-cloud-part-4-immutable-servers, Junio 2013. (Visitado el 20/10/2015).

[24] Melvin Conway. Conway's law. http://www.melconway.com/Home/Conways_Law.html, Abril 1968. (Visitado el 16/06/2015).

[25] Michael DeHaan. Immutable infrastructure is the future. http://michaeldehaan.net/post/118717252307/immutable-infrastructure-is-the-future. (Visitado el 13/08/2015).

[26] Phil Dibowitz. Scaling systems configuration at facebook. https://www.youtube.com/watch?v=SYZ2GzYAw_Q, 2013. (Visitado el 01/06/2015).

[27] Docker. Docker frequently asked questions (faq). https://docs.docker.com/engine/misc/faq/#what-does-docker-add-to-just-plain-lxc. (Visitado el 26/10/2015).

[28] Aliza Earnshaw. What is a devops engineer? https://puppetlabs.com/blog/what-is-a-devops-engineer, Mayo 2013. (Visitado el 25/10/2015).

[29] Damon Edwards. The (short) history of devops. http://itrevolution.com/the-history-of-devops/. (Visitado el 25/10/2015).

[30] Red Hat. Introduction to control groups (cgroups). https://access.redhat.com/articles/1353593. (Visitado el 26/10/2015).

[31] Red Hat. Introduction to linux containers. https://access.redhat.com/articles/1353593, Agosto 2015. (Visitado el 25/10/2015).

[32] Jeffrey Hulten. Using kanban and chef. http://devtalks.io/talks/chefconf-san-francisco/2013/using-kanban-and-chef-a-case-study.html#.VWzJgel7h5R, 2013. (Visitado el 01/06/2015).

[33] Adam Jacob. Chef style devops kungfu - chefconf 2015 - youtube. https://www.youtube.com/watch?v=_DEToXsgrPc, 2015. (Visitado el 29/05/2015).

[34] Gene Kim. Devops distilled, part 1: The three underlying principles. http://www.ibm.com/developerworks/library/se-devops/part1/, Enero 2013. (Visitado el 25/10/2015).

[35] Roman Komkov. Ansible at glogster. http://www.slideshare.net/decay_of_mind/ansible-at-glogster. (Visitado el 27/10/2015).

[36] Emilio Lorenzón. *Sistemas y Organizaciones - Parte II*. 2011.

[37] Tom Poppendieck Mary Poppendieck. *Lean Software Development: An Agile Toolkit*. Addison Wesley, 2004.

[38] Paul Menage. Cgroups. https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt. (Visitado el 26/10/2015).

[39] Eric W. Biederman Michael Kerrisk. Linux programmer's manual - namespaces. http://man7.org/linux/man-pages/man7/namespaces.7.html. (Visitado el 26/10/2015).

[40] Michael Warrilow Thomas J. Bittman, Philip Dawson. Magic quadrant for x86 server virtualization infrastructure. http://www.gartner.com/technology/reprints.do?id=1-2JGMVZX&ct=150715&st=sb, Julio 2015. (Visitado el 25/10/2015).

[41] Toyota. Just-in-time — philosophy of complete elimination of waste. http://www.toyota-global.com/company/vision_philosophy/toyota_production_system/just-in-time.html. (Visitado el 06/07/2015).

[42] Chris Sexsmith, Trevor A. Roberts Jr., Yvo van Doorn, Egle Sigler, Josh Atwell. *DevOps for VMware Administrators*. VMware Press, Abril 2015.

[43] Pablo Wright. Implementing puppet at a south american government agency, challenges and solutions - youtube. https://www.youtube.com/watch?v=Uqgs-XF_Uqg, 2014. (Visitado el 30/05/2015).

[44] Kieran Lal y Gerhard Killesreiter. An overview of the drupal infrastructure and plans for future growth. https://www.drupal.org/files/issues/Drupal.org-Architecture.pdf, 2006. (Visitado el 16/06/2015).

[45] Eliyahu M. Goldratt y Jeff Cox. *The Goal*. North River Press, 1984.

[46] Adam Jacob y otros. Chef style devops kung fu. https://github.com/chef/devops-kungfu. (Visitado el 26/10/2015).

Glosario

Ansible es una plataforma de software libre para configurar y administrar computadoras. 39, 43, 45, 46, 75

Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. 12

Debian es un sistema operativo libre GNU/Linux, desarrollado por miles de voluntarios alrededor del mundo, que colaboran a través de Internet. 13, 52

Docker Hub es un repositorio de imágenes de contenedores público donde se pueden descargar imágenes. 51

GIT es un sistema de control de versiones distribuido y de código abierto que permite manejar proyectos de cualquier tamaño de forma muy eficiente. 67

Gitlab es un manejador de repositorios Git basado en web con manejo de wiki y seguimiento de tickets. 39

Guaraní es un sistema para la gestión académica implementado por el SIU. 13

ITIL es un acrónimo inglés de Information Technology Infrastructure Library. Es un conjunto de conceptos y buenas prácticas para la gestión de servicios de tecnologías de la información, el desarrollo de tecnologías de la información y las operaciones relacionadas con la misma en general. 7

LaTeX es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Por sus características y posibilidades, es usado de forma especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas. 73, 89, 90

Mapuche es un sistema de recursos humanos implementado por el SIU. 13

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario. MySQL AB es la compañía que lo desarrolla como software libre en un esquema de licencia dual. 12, 13, 51

Nginx se trata de un servidor web liviano, orientado a obtener un rendimiento muy elevado. 12, 54

PDF es un formato de almacenamiento para documentos digitales independiente de plataformas de software o hardware. 89, 90

PHP PHP es un lenguaje de programación de propósito general e interpretado que es particularmente utilizado para el desarrollo de sitios web. 51, 52

Packer es una herramienta de software libre para la creación de imágenes de máquinas a partir de una configuración. 39

Pilagá es un sistema web de gestión presupuestaria, financiera y contable implementado por el SIU. 13, 44

Playbook un playbook es un archivo de texto plano que constituye el lenguaje de configuración, despliegue y orquestación de Ansible. Pueden definir una política que se quiere asegurar en los sistemas remotos o una serie de pasos en general. 39, 43, 44, 46–50, 55, 59, 91–93

PostgreSQL es un Sistema de gestión de bases de datos relacional orientado a objetos y libre, publicado bajo la licencia PosgreSQL, similar a la BSD o la MIT. 12, 13, 44

SIU-Toba es un Ambiente de Desarrollo Web que permite construir aplicaciones transaccionales. Esta herramienta de desarrollo rápido posee una arquitectura basada en componentes y una IDE de edición propia. Creado por SIU y con licencia de software libre. 13, 44

SIU es un acrónimo de Sistema de Información Universitaria, el cual desarrolla soluciones informáticas y brinda servicios para el Sistema Universitario Nacional y distintos organismos de gobierno. El SIU depende del Consejo Interuniversitario Nacional. II, III, V, 13, 15, 44–46, 51

Squeeze es una versión del sistema operativo Debian 6.0 que fue lanzada el 6 de febrero de 2011. 13

Tomcat funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation. 12

Ubuntu es un sistema operativo basado en GNU/Linux y que se distribuye como software libre. 13

VMware (VM de Virtual Machine) es una filial de EMC Corporation que proporciona software de virtualización disponible para ordenadores compatibles X86. 12, 43

Vagrant es una herramienta para la creación y configuración de entornos virtualizados. 39, 43

Virtualbox es una plataforma de virtualización de código abierto. 39

Windows es el nombre de una familia de distribuciones de software para PC, smartphone, servidores y sistemas empotrados, desarrollados y vendidos por Microsoft. 13

Work-in-Progress abreviado como WiP, Work in Progress, que en español se traduce como trabajo en progreso, hace referencia precisamente al trabajo que en un determinado momento se está llevando adelante. 21

YAML es un acrónimo recursivo que significa "YAML no es otro lenguaje de marcado". Es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado por el RFC 2822. 91, 92

fragile box Es un término utilizado por Nassim Nicholas Taleb en su libro Antifragile. Se utiliza para describir un servidor que corre un software que se ha vuelto muy crítico y nadie quiere tocar por miedo que deje de funcionar. 15

hosts virtuales un host virtual es, en el contexto de un servidor web, un mecanismo que permite ejecutar varios sitios web en el mismo servidor. 51, 54

roles un rol en Ansible permite agrupar contenido de manera que sea más simple reusar y compartir soluciones con otros usuarios. 46

sistema operativo host en el contexto de la virtualización, se entiende por sistema operativo host a aquel en el cuál se ejecutan los servidores virtuales o los contenedores. En otras palabras, puede verse como el sistema operativo de base. 81–83, 85

snapshot se entiende por Snapshot o imagen instantánea de una máquina virtual a la captura de su estado y datos en un punto específico en el tiempo. Así, un snapshot permite volver hacia atrás todos los cambios realizados en una máquina virtual hasta un instante de tiempo conocido. 7, 14, 59

ssh Secure SHell es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. 91

wiki es una plataforma web colaborativa que permite compartir contenido de forma rápida y sencilla y que es editable a través de un navegador. Las wikis además permiten ver los cambios que se han ido realizando en cada uno de sus artículos junto con la información del autor de cada cambio. 69